MASTER EN SCIENCES INFORMATIQUES

Proposition d'une démarche d'analyse intégrant IDA et memo-Proges

Delor, Emmanuelle; Schayes, Marie-Claire

Award date:
1984

Awarding institution:
Universite de Namur

Facultés Universitaires Notre-Dame de la Paix (Namur)
Institut d'Informatique

PROPOSITION

D'UNE DEMARCHE D'ANALYSE

INTEGRANT IDA ET MEMO-PROGES.

Promoteur : Mr R. LESUISSE

Mémoire présenté par

Emmanuelle DELOR
Marie-Claire SCHAYES

en vue de l'obtention
du grade de
Licencié et Maître en Informatique.

Année académique 1983-1984.

E. DELOR, M-C. SCHAYES - Proposition d'une démarche d'analyse intégrant IDA et MEMO-PROGES.

- ERRATA -

PLAN, *dans le point* 3.3.2.2.3. ajouter C. Spécification des modules d'entrée-sortie.

page 15, ligne 15 lire : adoptant

page 19, (1) lire : annexe 3

page 28, ligne 25 lire : § 2.2.2.1.1.

page 37, ligne 16 lire : figure 2.6.

page 38, dernière ligne . une erreur fatale est une erreur qui compromet irrémédiablement la suite de l'exécution du programme et en provoque l'arrêt immédiat

. une erreur non fatale est définie a contrario

page 40, au lieu de figure 2.5 lire : figure 2.10.

page 55, ligne 22 lire figure 3.2.

page 82, ligne 7 lire : processus

page 83, figure 3.23 Lire : modèle MEMO-PROGES de couplage des modules pour enchaîner les fonctions

figure 3.24 lire : modèle MEMO-PROGES de couplage des modules pour déclencher la première fonction

page 101, (2) Lire : annexe 3

page 109, dernière ligne lire : mono-file d'attente

BIBLIOGRAPHIE, à ajouter

[MYERS], Glenford J. MYERS : Reliable Software through composite design, Van Nostrand Reinhold Company, 1975.

[CLARINVAL.c], A. CLARINVAL : Indépendance de la structure des programmes par rapport à l'organisation des fichiers, Doument interne, Groupe S.

Nous tenons tout d'abord à remercier Monsieur Lesuisse pour toute l'aide et les encouragements qu'il n'a cessé de nous prodiguer en tant que directeur de notre mémoire et pour l'intérêt et l'attention qu'il a porté à ce travail.

Nous tenons tout spécialement à remercier Monsieur Clarinval pour tout ce que nous ont apporté ses enseignements, ses conseils, ses critiques constructives, les nombreuses discussions que nous avons pu avoir avec lui, et, surtout, pour sa constante disponibilité et son dévouement à notre égard.

Nous tenons également à remercier Monsieur Hennebert qui a bien voulu superviser notre travail au CIGER et guider nos premiers pas dans la réalisation de ce travail.

PLAN.

INTRODUCTION.

L'analyse d'un problème d'automatisation du traitement de l'information, depuis sa perception jusqu'à l'implantation de sa solution, couvre quatre étapes :

- l'étude d'opportunité qui propose un avant-projet de solution à partir des besoins exprimés par l'organisation,

- l'analyse conceptuelle qui, sur base de cet avant-projet de solution, élabore une solution conceptuelle indépendante de tout moyen de réalisation,

- l'analyse organique qui affine la solution conceptuelle retenue en vue d'obtenir une solution implémentable, et

- la réalisation qui implante la solution implémentable dans l'environnement de l'organisation en fonction des caractéristiques des moyens techniques (matériel et logiciel) disponibles.

La figure 0.1. présente le schéma du "cycle de vie" d'un projet informatique.

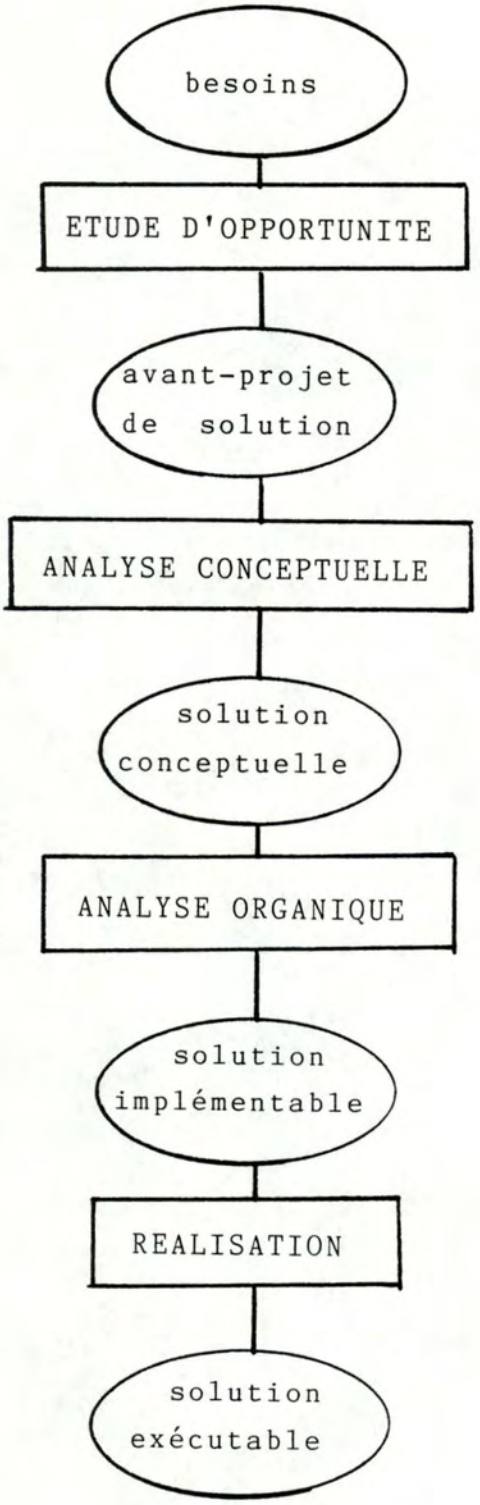


figure 0.1. "cycle de vie" d'un projet informatique.

Dans le cadre de ce mémoire, nous ne nous sommes intéressées qu'aux deux étapes intermédiaires de ce cycle de vie : l'étape d'analyse conceptuelle et l'étape d'analyse organique.

Notre objectif est de proposer une méthode d'analyse qui présente deux carctéristiques intéressantes :

- Elle s'appuie sur des modèles et outils existants, modèles et outils IDA (1) pour l'analyse conceptuelle, modèles et outils Memo-Proges (2) pour l'analyse organique. Ces modèles et outils ont été développés indépendamment les uns des autres.

- Elle met en oeuvre un véritable continuum entre l'analyse conceptuelle et l'analyse organique en suggérant une possibilité de construction systématique d'une solution implémentable à partir d'une solution conceptuelle.

---

(1) IDA (Interactive Design Approach) est un système-logiciel d'aide à la conception d'un système d'information, réalisé à l'Institut d'Informatique des Facultés Universitaires de Namur, en coopération avec le projet ISDOS. [Bodart-Pigneur].

(2) MEMO-PROGES (MEthode MOdulaire de PROgrammation pour l'informatique de GEStion) est une méthode d'analyse et de programmation développée par Clarinval. [Clarinval.a].

Toute méthode peut être caractérisée par le triplet (modèles, outils, démarche). En effet, une méthode doit s'appuyer sur des modèles; elle doit disposer d'outils et elle doit proposer une démarche, c'est-à-dire des règles de mise en oeuvre de ces modèles et outils.

Nous procéderons dès lors à la présentation de cette méthode en deux temps.

Dans un premier temps, nous exposerons les modèles et outils utilisés. Le chapitre I est consacré aux modèles et outils de l'analyse conceptuelle. Quant au chapitre II, il s'intéresse aux modèles et outils relatifs à l'étape d'analyse organique.

Dans un second temps, nous exposerons la démarche que nous avons adoptée pour réaliser les deux étapes d'analyse conceptuelle et d'analyse organique. Cette démarche se différenciera selon le type de problème traité. Le chapitre III sera consacré aux problèmes avec flux d'information. Quant au chapitre IV, il s'intéressera aux problèmes sans flux d'information.

Le chapitre V discutera de la réalisation en Memo-Proges de la solution obtenue au terme de cette démarche dans différents modes d'exploitation : le mode batch et le mode interactif multi-utilisateurs.

CHAPITRE I - MODELES ET OUTILS DE L'ANALYSE CONCEPTUELLE.

1.0. INTRODUCTION.

Ce chapitre est consacré à l'étape d'analyse conceptuelle. Il a pour objectif de proposer une façon générale de poser un problème d'automatisation du traitement de l'information. A cet effet, dans un premier temps, nous exposerons le modèle général de spécification d'un problème et son contenu. Dans un second temps, nous l'appliquerons à 2 cas d'application (1), représentatifs chacun d'un type de problème : le traitement des commandes-clients de la firme Petitpas, représentant un problème avec flux d'information et la gestion du fichier des anciens de l'Institut, représentant un problème sans flux d'information. Nous expliquerons en quoi ces problèmes diffèrent et comment on peut leur appliquer le modèle général proposé.

1.1. ROLE DE L'ANALYSE CONCEPTUELLE.

L'analyse conceptuelle couvre la deuxième étape dans le cycle de vie du développement d'un projet informatique. Elle consiste à spécifier le problème en fournissant à partir de l'avant-projet de solution dégagé par l'étude d'opportunité, une solution conceptuelle sous forme d'un schéma conceptuel. Par schéma conceptuel, on entend une description en termes de concepts et exprimée de préférence à l'aide d'un langage donné (2)

- des informations,
- des traitements qui y sont associés, et
- des conditions d'activation de ces derniers,

qu'il faudra représenter dans le futur système d'information (S.I.) pour qu'il satisfasse les besoins exprimés dans l'organisation, indépendamment des moyens de réalisation, [Bodart-Pigneur].

---

(1) L'énoncé de ces 2 cas d'application est donné dans les annexes 2&3.
(2) La méthode de Bodart contient la définition d'un langage formel de spécification, appelé D.S.L. dont la définition est donnée dans l'annexe 1.

1.2. MODELE GENERAL DE SPECIFICATION D'UN PROBLEME.

1.2.1. OBJECTIFS.

Le rôle de l'analyse conceptuelle étant de spécifier un problème, nous allons dans ce paragraphe proposer une façon générale de poser un problème d'automatisation du traitement de l'information.
Spécifier un problème, c'est découvrir et décrire d'une part les "fonctions" de traitement de l'information qu'il présente et d'autre part les objets sur lesquels agissent ces fonctions.

1.2.2. CONTENU.

1.2.2.1. Concept de fonction.

Par fonction, nous entendons un traitement élémentaire qui a pour objectif
soit la production de messages-résultats
soit une action élémentaire sur la mémoire du S.I..
Au concept "fonction" est dès lors associé un ensemble de primitives de traitement de messages et d'action sur la mémoire du S.I.. Une fonction réalisant une sémantique simple, elle doit pouvoir s'exprimer avec un seul verbe ou substantif verbal correspondant à cet objectif élémentaire. Toute fonction pourra dès lors être exprimée au moyen des verbes suivants, sans que leur liste soit exhaustive :

éditer ‹message›
afficher ‹message›
afficher ‹mémoire›
modifier ‹mémoire›
ajouter ‹mémoire›
supprimer ‹mémoire›

[Bodart-Lesuisse].

1.2.2.2. Objets : mémoire et messages.

De la définition de fonction, on s'aperçoit que les objets sur lesquels agissent les fonctions se répartissent en deux catégories :

- la mémoire du système et
- les messages.

La mémoire du S.I. reprend toutes les informations stockées dans le système. Elle sera modélisée à l'aide du modèle Entité-Association (E-A), [Chen], dont nous nous bornons ici à rappeler les concepts fondamentaux. Selon ce modèle, les informations sont structurées en entités, associations et attributs sur lesquels peuvent agir des contraintes d'intégrité.

- Entité : N'importe quel objet concret ou abstrait du monde réel peut constituer une entité. Une entité n'existe en tant que telle que par rapport à un individu ou à un groupe qui la considère comme un tout et lui confère une existence autonome.
- Association : Une association est une relation qu'un individu ou un groupe perçoit entre deux ou plusieurs entités où chacune assume un rôle donné. L'existence d'une association est contingente à l'existence des entités qu'elle unit.
- Attribut : Un attribut est une caractéristique ou une qualité attribuée par un individu ou un groupe à une entité ou une association. L'existence d'un attribut est contingente à l'existence de l'entité ou association concernée .
- Contrainte d'intégrité : Une contrainte d'intégrité est un attribut, non représenté par les concepts de base, que doivent satisfaire les informations appartenant à la mémoire du S.I.. Parmi les contraintes d'intégrité les plus fréquemment rencontrées, on peut citer les contraintes de connectivité, d'existence, de format, de valeurs ...

Quant aux messages, second type d'objets sur lesquels agissent les fonctions, ils sont les véhicules des informations stockées dans la mémoire du S.I. ou dérivées de celles-ci. Ces informations sont échangées
- soit entre le S.I. et son environnement et inversément
- soit à l'intérieur du S.I..

Par rapport aux traitements, on distingue donc 3 types de messages :
- message-entrée : en provenance de l'environnement et à destination d'une fonction du S.I..
- message-sortie : émis par une fonction du S.I. et à destination de l'environnement.
- message-interne : échangé entre des fonctions du S.I..

Les messages étant composés directement ou indirectement (données calculées ou dérivées) d'éléments de la mémoire du S.I., seront décrits et définis en termes de ces éléments.

Spécifier les informations revient donc à caractériser, en D.S.L., tout message, toute entité, toute association et tout attribut.

1.2.2.3. Schéma de synthèse du modèle général.

Le modèle général de spécification que nous venons de décrire peut être schématisé par la figure 1.1.

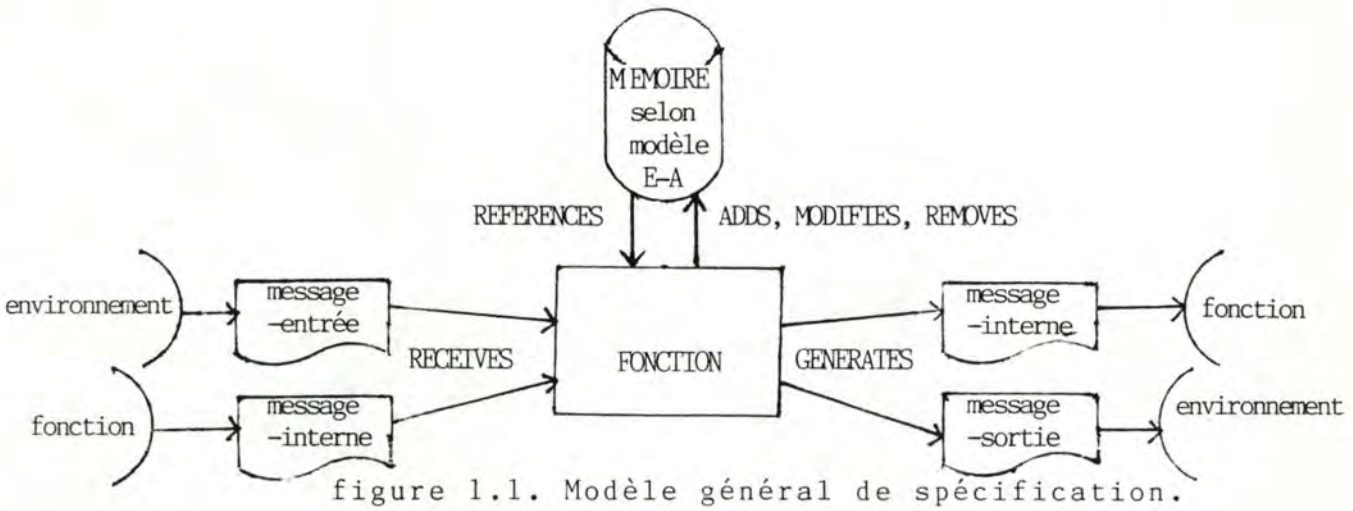


figure 1.1. Modèle général de spécification.

Spécifier un problème, c'est donc donner pour chaque fonction :

- une description de la situation initiale, c'est-à-dire une description du message à l'entrée de la fonction.
- une description de la situation finale, c'est-à-dire une description du message à la sortie de la fonction.
- une description de la mémoire nécessaire pour résoudre le problème.
- une description de l'objectif à réaliser et des performances souhaitées.

Soit à spécifier, par exemple, la fonction "afficher ‹résultat-contrôle-identité›":

Représentation en langue naturelle	Représentation dans la langue formelle D.S.L.
Fonction : afficher ‹résultat-contrôle-identité›	DEFINE PROCESS afficher-résultat-contrôle-identité;
Message-entrée : cmde-en-tête	RECEIVES cmde-en-tête;
Message-sortie : cmde-en-tête-marquée	GENERATES cmde-en-tête-marquée;
Mémoire : client	USES client;
Objectif : Vérifier que l'en-tête du bon de commande est correcte	DESCRIPTION; Cette fonction vérifie que l'en-tête du bon de commande est correcte.
Performances : Rejeter le moins de commandes possible	On veillera à rejeter le moins de bons de commande possible;

1.3. SITUATIONS DE CONCEPTION.

Les problèmes structurés de gestion en informatique (1) peuvent se diviser en 2 classes : la classe des problèmes avec flux d'information qui recouvre l'ensemble des problèmes opérationnels classiques (gestion des commandes, calcul des salaires, ...) et la classe des problèmes sans flux d'information, problèmes orientés données (mise à jour d'un signalétique client avec production de documents statistiques, ...).
Après avoir défini et caractérisé ces 2 classes de problèmes, nous leur appliquerons le modèle général proposé.

(1) Dans ce mémoire, nous nous sommes limitées à ce type de problèmes.

1.3.1. PROBLEMES AVEC FLUX D'INFORMATION.

1.3.1.1. Définition et caractéristiques.

Par problème avec flux d'information, nous entendons un problème pour lequel, du point de vue externe, une même suite de transformations est appliquée à chaque occurrence d'un type de message d'entrée afin d'obtenir une occurrence d'un type de message de sortie. Il peut dès lors être modélisé par la figure 1.2.

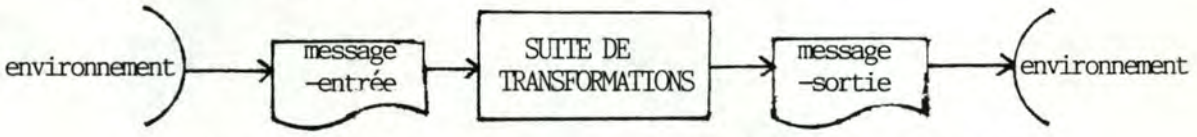


figure 1.2. Problème avec flux d'information.

Le traitement des commandes-clients de la firme Petitpas (1) est un problème avec flux d'information. En effet, le bon de commande émis par le client va subir, dans l'organisation, un certain nombre de transformations (vérification, enregistrement, création expédition, facturation, ...) avant de devenir une facture ou un justificatif de non livraison à destination du client.

Trois faits caractérisent un problème avec flux d'information :

- Dans un problème avec flux, les messages passent par différents états (au message d'entrée correspond l'état initial, au message de sortie correspond l'état final). A chaque état du message (excepté l'état final) correspond une fonction lui permettant d'atteindre l'état suivant. Une fonction ne pourra dès lors être appliquée que si le message est dans l'état correspondant à cette fonction. Ainsi la fonction "éditer facture" ne pourra être appliquée que s'il y a eu "réquisition".
- L'ordre d'enchaînement des fonctions n'est pas quelconque.

(1) L'énoncé du cas Petitpas est donné dans l'annexe 2.

- Les fonctions sont soit déclenchées explicitement, soit déclenchées implicitement. La distinction entre déclenchement explicite et déclenchement implicite vient du rôle que joue l'utilisateur dans le système. Dans le déclenchement explicite, son rôle est actif : il appartient à l'utilisateur de décider du déclenchement d'une fonction et de la déclencher réellement quand il reçoit un message dans un certain état. Dans le déclenchement implicite, par contre, son rôle devient passif : il a donné la responsabilité au système de déclencher une fonction quand il reçoit un message dans un certain état.
  Dans un problème avec flux d'information, le déclenchement explicite d'une fonction est dû à la réception d'un message dans son état initial (message d'entrée). Quant au déclenchement implicite, il est dû à la réception d'un message dans un état intermédiaire (message interne), c'est-à-dire un état autre que l'état initial et que l'état final.

1.3.1.2. Application du modèle général.

Le modèle général de spécification comporte 2 volets :
- l'un relatif aux traitements : l'identification des fonctions et leur enchaînement;
- l'autre relatif aux informations : l'identification des messages et des informations qui constituent la mémoire du système.

1.3.1.2.1. Identification des fonctions.

1° Découpe du flux.

Dans un problème avec flux d'information, le concept de fonction n'est pas immédiat. Il faut avant d'y arriver procéder à une découpe du flux. Cette découpe ne se fera pas de manière anarchique, mais en fonction d'un repère appelé phase, [Bodart-Pigneur].

Par définition, une phase est un traitement (manuel ou automatisable) possédant une unité spatio-temporelle d'exécution. Cette unité d'exécution implique que la phase soit entièrement exécutée dans une cellule d'activités, c'est-à-dire un centre d'activité homogène dans le temps et dans l'espace, doté de ressources humaines et/ou matérielles et pourvu de règles de comportement nécessaires à son fonctionnement. De la définition, on tire 2 critères organisationnels d'identification d'une phase : l'unité spatiale d'exécution des traitements d'une part, et l'unité temporelle d'exécution des traitements d'autre part. Par unité spatiale d'exécution des traitements, on entend l'absence de changement spatial dans l'organisation et l'absence de changement de ressources. Par unité temporelle, on entend que

- le déroulement de l'exécution d'une phase puisse se faire sans interruption (absence de point d'attente, de point de décision humaine);
- la phase ait la même fréquence d'acquisition des messages d'entrée;
- tous les traitements qui constituent la phase aient la même périodicité d'exécution.

En utilisant le concept de phase, on procède donc à la découpe du flux en fonction de la double dimension espace-temps.

Dans notre cas d'application, le flux relatif aux commandes-clients, situé entre la réception des commandes et l'émission des documents d'expédition à destination du client, peut être découpé en les 9 phases suivantes :

PHASES	CRITERE D'IDENTIFICATION
1. PREPARATION BON DE COMMANDE traitement manuel : décacheter les enveloppes et vérifier que les bons de commande sont signés.	Point d'attente et de décision. Changement de ressources.

2. ENREGISTREMENT BON DE COMMANDE CLIENT

Traitement interactif : les opérateurs d'enregistrement procèdent via des terminaux, à l'identification du client d'un bon de commande signé et à l'enregistrement des lignes de commande correctes.

Point de décision.
Changement de ressources.

3. MISE A JOUR MOINS DU STOCK

Traitement automatisable : mise à jour des stocks relatifs à une commande enregistrée. La partie livrable donne lieu à un bon de réquisition; le reste est mémorisé et donnera lieu à une ou plusieurs livraisons différées.

Point d'attente.

4. ORDONNANCEMENT

Traitement automatisable : lorsque n bons de réquisition ont été émis, on procède à un ordonnancement de façon à optimiser le parcours du magasinier.

Changement spatial.
Changement de ressources.

5. PARCOURS DU MAGASIN

Traitement manuel : prélever les produits.

Point d'attente.
Changement de ressources.

6. CONSTITUTION DU COLIS

Traitement manuel : au terme du parcours par le magasinier, les opérateurs d'emballage constituent les colis grâce aux bons de réquisition.

Point d'attente et de décision.
Changement de ressources.

7. CORRECTION DU SYSTEME

Traitement interactif : si le colis n'a pu être complètement reconstitué, on procède via des terminaux à toutes les corrections nécessaires dans le système pour tenir compte de ses incompatibilités.

8. FACTURATION

Traitement automatisable : impression, pour une expédition donnée, du bon de livraison et de ses documents d'accompagnement.

Différence de périodicité.

9. ARCHIVAGE

Traitement automatisable : archiver les commandes dont les quantités-dues sont toutes nulles.

Les phases une fois identifiées, peuvent être décomposées en fonctions selon les règles définies au § 1.2.2.1..

Ainsi dans la phase "ENREGISTREMENT BON DE COMMANDE CLIENT", on distingue les fonctions suivantes :

afficher‹résultat-contrôle-identité›
afficher‹résultat-contrôle-ligne›
afficher‹résultat-contrôle-total›
ajouter‹client›
modifier‹client›
ajouter‹ligne-commande›
éditer‹justificatif-refus›

2° Coordination.

Dans un problème avec flux d'information, l'ordre d'enchaînement des fonctions n'est pas quelconque. La décomposition en fonctions étant réalisée, il faut à présent décrire les règles d'enchaînement de ces fonctions. Ces règles expriment soit la logique du problème, soit des contraintes organisationnelles. Nous avons choisi pour représenter les conditions de déclenchement et d'enchaînement des fonctions le modèle de la dynamique décrit dans [Bodart-Pigneur]. Dans ce modèle, on trouve 3 concepts que nous allons rapidement définir : celui de processus, celui d'événement et celui de point de synchronisation.

- Processus : Un processus est l'exécution d'une procédure de traitement de l'information dont la progression peut être représentée, à des points discrets dans le temps, par son état. Les états remarquables d'un processus sont l'attente, l'état actif et l'état terminé.
- Evénement : Un événement correspond à un changement d'état du S.I. localisé dans le temps et dans l'espace. Un événement est dit externe s'il correspond à un franchissement de la frontière du S.I. par un message en provenance de son environnement et qui déclenche en réaction, un processus du S.I. Un événement est dit interne s'il

correspond à un changement interne au S.I., tel que le changement d'état d'un processus ou la réalisation d'un point de synchronisation.

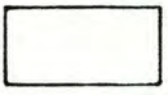
- Point de synchronisation : Un point de synchronisation est un mécanisme de coordination d'événements.

La base de la dynamique est l'événement en ce sens que l'occurrence d'un événement peut causer 2 actions dynamiques : soit le déclenchement d'un processus (ce déclenchement pourra éventuellement être conditionnel ou multiple);
soit la contribution à un point de synchronisation (cette contribution pourra éventuellement être conditionnelle).

En appliquant ce modèle à la phase "ENREGISTREMENT BON DE COMMANDE CLIENT" et en adaptant les conventions graphiques (1), on obtient le schéma représenté à la figure 1.3..

---

(1) Les conventions graphiques du schéma de la dynamique sont les suivantes : - un processus



- un événement

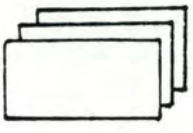
- une relation de déclenchement

- une condition

- un point de synchronisation

- un déclenchement multiple

- une accumulation



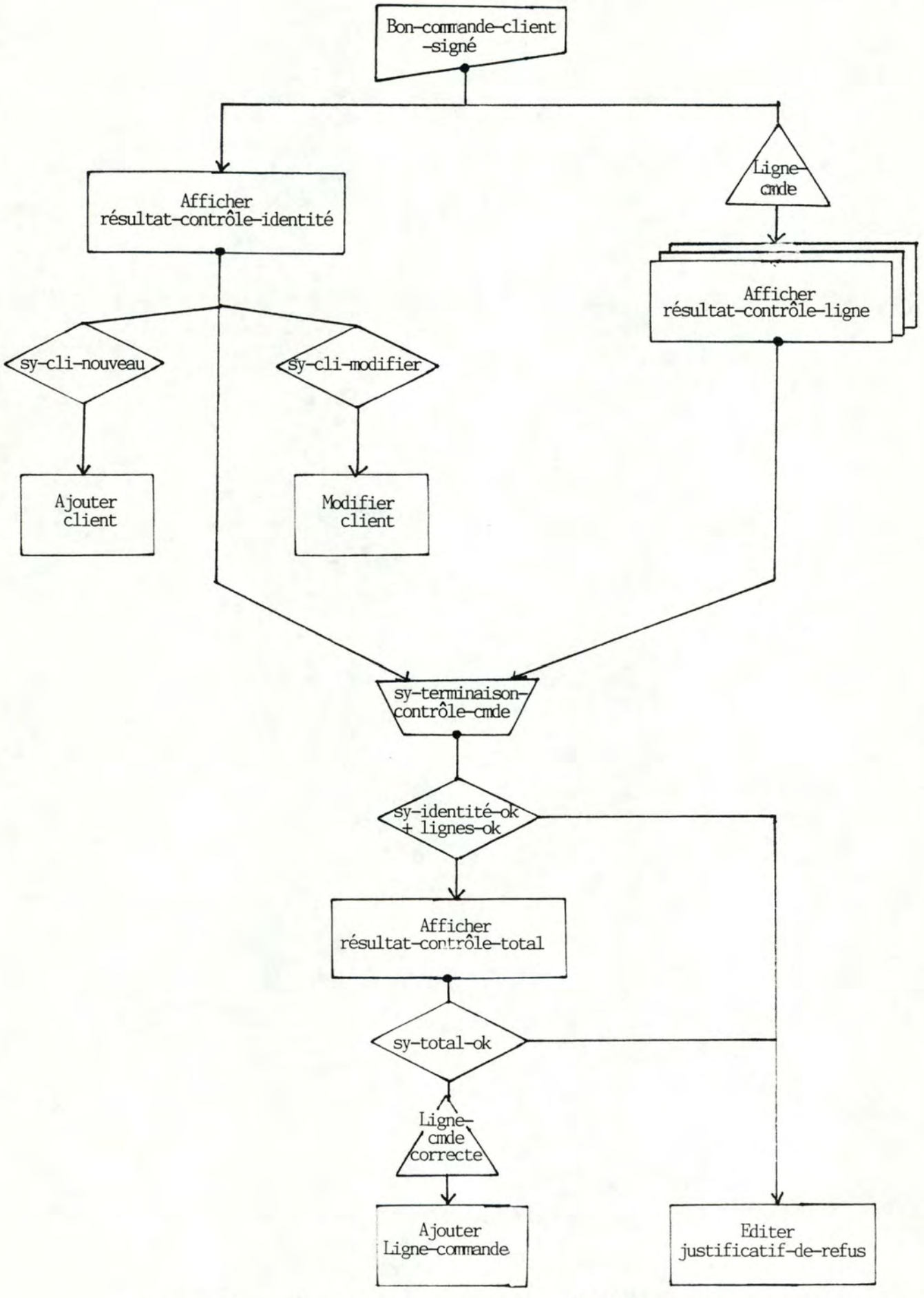


figure 1.3. Dynamique de la phase "ENREGISTREMENT-BON-DE-COMMANDE-CLIENT".

1.3.1.2.2. Identification de la mémoire et des messages.

Au niveau de la découverte des fonctions, la phase a joué un rôle primordial. Elle va également jouer un rôle central dans la découverte des données. Lieu d'identification des fonctions, elle sera aussi un lieu d'identification des données. C'est en effet au niveau de la phase que l'on sera désireux de percevoir les structures d'informations sur lesquelles agissent les fonctions. On a distingué 2 types d'informations : la mémoire et les messages.

Ainsi, au niveau de la phase "ENREGISTREMENT BON DE COMMANDE CLIENT", on distingue la structure d'informations suivante :

1. MESSAGES
   - bon-de-commande-signé
   - justificatif-refus
   - résultat-contrôle-identité
   - résultat-contrôle-ligne
   - résultat-contrôle-total
2. MEMOIRE ou sous-schéma conceptuel modélisé à l'aide du modèle Entité-Association. L'expression graphique de ce sous-schéma est présentée à la figure 1.4. (1)

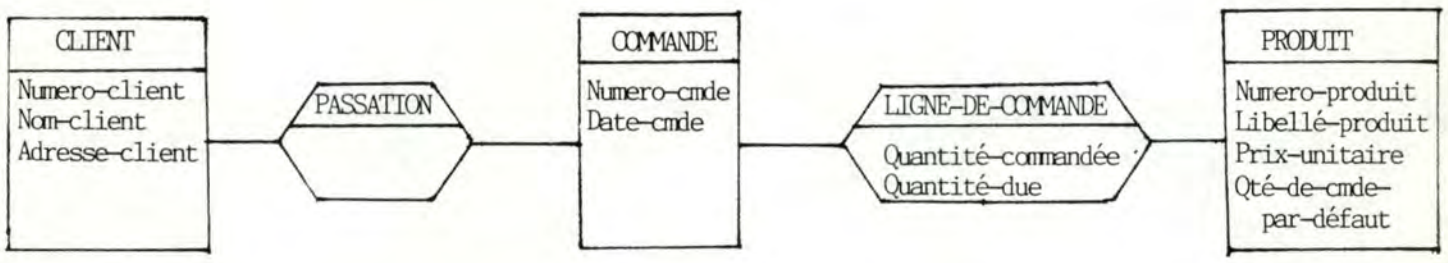


figure 1.4. Sous-schéma conceptuel de la phase "ENREGISTREMENT BON DE COMMANDE CLIENT".

(1) Les rectangles représentent les entités
Les hexagones représentent les associations.

1.3.2. PROBLEMES SANS FLUX D'INFORMATION.

1.3.2.1. Définition et caractéristiques.

Les problèmes sans flux d'information sont définis *a contrario* des problèmes avec flux d'information. Il n'y a donc pas dans un problème sans flux d'information de suite de transformations appliquée à un message d'entrée le faisant passer par différents états intermédiaires avant d'obtenir le message de sortie.
Par problème sans flux d'information, nous entendons une structure passive d'informations sur laquelle agissent des fonctions.
Un problème sans flux d'information pourra dès lors être modélisé par la figure 1.5..

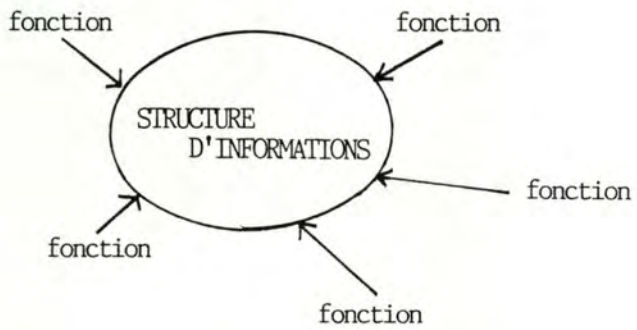


figure 1.5. Problème sans flux d'information.

Trois faits caractérisent un problème sans flux d'information :
- les fonctions sont indépendantes les unes des autres;
- elles doivent être déclenchées explicitement;
- chaque traitement part d'un message d'entrée
  - soit pour mettre à jour la structure d'informations,
  - soit pour en extraire une partie,

  sans nécessiter d'état intermédiaire du message entre son état initial et l'action ou l'extraction sur ou à partir de la structure d'informations.

Avec les problèmes sans flux d'information, on est très proche de la notion de "type abstrait" [Liskov]. En effet, par type abstrait, on entend une structure passive d'informations associée à une conception de base du problème entièrement caractérisée par

- les propriétés inhérentes à cette structure;
- l'identification et la spécification des opérations (composants actifs du type) que l'on peut effectuer sur cette structure et qui lui sont biunivoquement associées.

1.3.2.2. Application du modèle général.

Dans un problème avec flux d'information, on a commencé par structurer les traitements avant de structurer les données. Dans un problème sans flux d'information, l'ordre sera inversé, car c'est un problème qui se définit plus par les données que par les traitements. Contrairement à un problème avec flux d'information, l'application du modèle général (§ 1.2.) est immédiate.

1.3.2.2.1. Identification de la mémoire et des messages.

De l'énoncé du problème de la gestion du fichier des anciens étudiants de l'Institut (1), on dégage quasi-immédiatement la structure d'informations suivante où l'on retrouve la distinction entre messages et mémoire :

1. MESSAGES
   - ancien-à-ajouter
   - ancien-à-modifier
   - ancien-à-supprimer
   - sélection-ancien-à-afficher
   - sélection-étiquettes-ancien
   - sélection-liste-alphabétique
   - sélection-tableau-récapitulatif

(1) L'énoncé de ce problème est donné dans l'annexe 4.

ancien-affiché
étiquettes-ancien
liste-alphabétique
tableau-récapitulatif

2. MEMOIRE modélisée à l'aide du modèle Entité-Association et représentée à la figure 1.6.

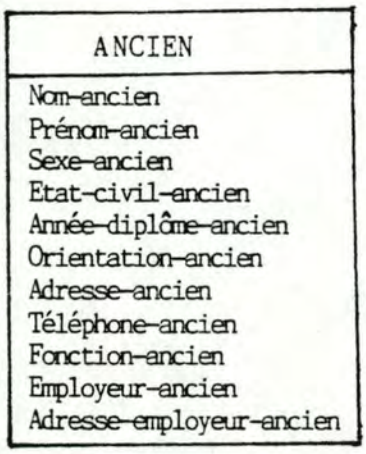


figure 1.6. Schéma conceptuel du problème "ANCIEN".

1.3.2.2.2. Identification des fonctions.

De cette structure d'informations et de l'énoncé, on distingue tout aussi immédiatement les fonctions suivantes :

ajouter ‹ancien›
modifier ‹ancien›
supprimer ‹ancien›
afficher ‹ancien›
éditer ‹étiquettes-ancien›
éditer ‹liste-alphabétique›
éditer ‹tableau-récapitulatif›

L'exécution de ces fonctions se fait indépendamment les unes des autres.

1.4. CONCLUSION.

Nous venons de montrer qu'il est possible de spécifier n'importe quel problème d'automatisation du traitement de l'information à

partir des 3 modèles présentés dans [Bodart-Pigneur]. Ces modèles sont : - le modèle Entité-Association (E-A),
- le modèle de structuration des traitements, et
- le modèle de la dynamique.

Au terme de l'analyse conceptuelle, on obtient
- un modèle des informations exprimé à l'aide du modèle E-A
- un modèle des traitements qui est
  soit, pour un problème avec flux d'information, un réseau de fonctions communiquant par l'intermédiaire de messages auquel on a surimposé des contraintes dynamiques (contraintes de déclenchement, point de synchronisation, ...), expressions de règles organisationnelles;
  soit, pour un problème sans flux d'information, un catalogue de fonctions.

Ces 2 modèles peuvent être exprimés dans le langage D.S.L. (1).

(1) La définition du langage D.S.L. est donnée dans l'annexe 1.

CHAPITRE II - MODELES ET OUTILS DE L'ANALYSE ORGANIQUE.

2.0. INTRODUCTION.

Ce chapitre poursuit un double objectif :
- présenter les modèles Memo-Proges en tant que réinterprétation des modèles de l'analyse conceptuelle;
- présenter l'outil Memo-Proges en tant que méthode de programmation.

Nous verrons comment les concepts d'entité et d'association de l'analyse conceptuelle se transformeront dans la première étape en fichiers logiques et dans la seconde étape en fichiers physiques.
Les fonctions et les mécanismes de la dynamique, quant à eux, seront transposés au niveau logique en modules qui seront implémentés physiquement sous forme de sous-routines.

2.1. ROLE DE L'ANALYSE ORGANIQUE.

L'analyse organique définit dans un premier temps, une architecture logique des traitements, interprétation opérationnelle du modèle conceptuel des traitements, et une architecture logique des données représentant les informations. Dans un second temps, elle définit la réalisation physique de ces 2 architectures.

L'architecture logique des traitements consiste en une identification et une spécification des modules. Ces modules interagissent les uns avec les autres au moyen d'interfaces.
L'architecture logique des données consiste en la définition des "fichiers logiques" (1).
Ces 2 architectures sont indépendantes de l'environnement de développement du logiciel (hardware et software disponibles). Elles ne tiennent compte d'aucune considération d'implémentation, mais répondent seulement à des exigences définies lors de l'analyse conceptuelle. Les modules et les fichiers logiques

(1) Le concept de fichier logique sera défini en temps opportun.

sont conçus sans aucune recherche d'efficacité, de performances ou d'optimisation.

La réalisation physique des traitements consiste à structurer et programmer les modules de l'architecture logique des traitements. La réalisation physique des données concrétise les fichiers logiques en fichiers physiques.

2.2. ARCHITECTURE LOGIQUE DU SYSTEME.

Construire l'architecture logique du système, c'est identifier et spécifier les composants organiques

- relatifs aux données, c'est-à-dire le fichier logique;
- relatifs aux traitements, c'est-à-dire les modules et les interfaces.

La construction de cette architecture sera guidée par le fait que nous avons voulu la considérer comme une réinterprétation organique des éléments du modèle conceptuel.
Dès lors, pour chaque composant organique, nous adopterons le plan suivant :
- Définition des composants organiques;
- Règles d'identification des composants par réinterprétation des éléments conceptuels;
- Modèle de spécification.

2.2.1. ARCHITECTURE LOGIQUE DES DONNEES.

L'architecture logique des données consiste à structurer les informations en termes de fichiers logiques.

2.2.1.1. Fichier logique.

1° Définition.

Par fichier logique (1), nous entendons un ensemble de données ayant, par rapport au traitement analysé (phase,fonction), même provenance ou même destination dans l'espace-temps environnant. Un élément de cet ensemble aura la structure d'un t-uple et sera appelé enregistrement.

---

(1) Le concept de fichier logique est emprunté à la méthode Memo-Proges.

2° Statuts des fichiers.

Memo-Proges définit 4 statuts de fichiers :

- fichier "Master"
- fichier "New"
- fichier "Alternate input"
- fichier "Updated"

Ces statuts sont définis par le mode d'intervention des fichiers dans les modules. L'intervention des fichiers dans les modules se définit selon 2 axes :

- le fichier intervient soit en entrée du module,
  soit en sortie du module.
- son mode d'accès logique est soit exhaustif (tous les enregistrements doivent être traités une et une seule fois),
  soit sélectif (tous les enregistrements ne doivent pas être traités ou certains doivent être traités plusieurs fois; à chaque accès, la partie du fichier à traiter doit donc être effectivement "sélectionnée").

Le tableau de la figure 2.1. représente les différents statuts de fichiers définis par les modes d'intervention.

Mode d'accès logique	EN ENTREE	EN SORTIE
MODE EXHAUSTIF	M fichier Master	N fichier New
MODE SELECTIF	A fichier Alternate input	U fichier Updated

figure 2.1. Statuts des fichiers.

Ce qui, en Memo-Proges se représente par la figure 2.2.

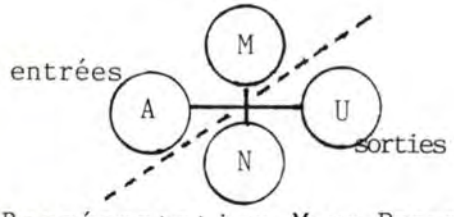


figure 2.2. Représentation Memo-Proges des statuts des fichiers.

Remarque : La définition d'un mode d'accès sélectif est déjà révélatrice d'un point de vue essentiel de la méthode : considérer des ensembles et non seulement des éléments. Ce point de vue aura des implications concrètes dans la structure interne des modules.

3° Opérations d'accès aux fichiers.

Les opérations d'accès possibles selon le statut du fichier sont présentées dans le tableau de la figure 2.3.

M	READ	N	WRITE
A	OBTAIN	U	INSERT REPLACE DELETE

figure 2.3. Opérations d'accès aux fichiers.

2.2.1.2. Transformation du modèle conceptuel des informations en fichiers logiques.

L'analyse conceptuelle a distingué 2 types d'informations :
- les messages et
- la mémoire modélisée à l'aide du modèle E-A (§ 1.2.2.2.).

A chaque ensemble de messages, entités et associations correspondra un fichier logique.

2.2.1.3. Spécification du fichier logique.

La spécification du fichier logique consiste en la description de chacun des types d'enregistrements de ce fichier.

2.2.2. ARCHITECTURE LOGIQUE DES TRAITEMENTS.

L'architecture logique des traitements consiste en une modularisation du système. Elle met donc en évidence deux composants organiques : le module et l'interface.

2.2.2.1. Module et Interface.

N'ayant pas d'accord parmi les auteurs sur la terminologie de ces 2 concepts, nous présenterons leurs définitions sous forme de discussion.

2.2.2.1.1. Module.

1° Définition.

Un module, au sens d'une méthode modulaire en général, est considéré comme une unité de conception, de validation et de maintenance. Pour le reste, les auteurs donnent quelques définitions plus précises.
Pour [Myers], un module est un groupe d'instructions dont on peut demander l'exécution de l'extérieur et qui a son propre ensemble de variables. La qualité d'un module se mesure par sa forte cohésion interne et son faible couplage externe, c'est-à-dire respectivement les relations entre les éléments d'un module et les relations entre les modules eux-mêmes. Dans tout programme, les éléments ont des relations. Il faut regrouper dans un module des éléments fortement liés, et mettre dans des modules différents des éléments non liés. D'autres qualités sont à observer telles que la simplicité du module, sa taille ...
[Parnas] quant à lui, ne donne pas de définition précise. Un module est essentiellement une unité de travail, une tâche (work assignment). Les modules doivent cacher de l'information. Chaque décision de conception susceptible d'être modifiée doit être enfermée dans un seul module qui en garde le secret. Les relations entre modules consistent uniquement en des hypothèses qu'ils font l'un par rapport à l'autre quant à leur façon de travailler.
Memo-Proges définit un module (1) comme étant en plus, une unité de compilation. Ceci signifie qu'un module Memo-Proges correspondra, au niveau de la réalisation physique, à un sous-programme COBOL.

2° Critère d'évaluation : la cohésion.

[Myers] définit différents types de cohésions internes, qu'il classe suivant une échelle de la plus faible à la plus forte.

---

(1) Dans la terminologie de Memo-Proges, un module est appelé une fonction. Afin d'éviter l'ambiguïté, avec l'analyse conceptuelle, nous utiliserons le terme de module.

La cohésion la plus faible, est la cohésion par coïncidence. Les éléments d'un module ne présentent aucune relation entre eux. Elle est souvent le résultat d'une découpe d'un programme en modules a posteriori.
La cohésion logique est celle où un module réalise plusieurs fonctions, et où les éléments de ce module ont entre eux seulement des relations logiques. A chaque appel, le module réalise une des fonctions d'une classe. Le choix de la fonction à exécuter est guidé par une information de contrôle qui aiguille vers la partie à exécuter.
La cohésion classique est la même que la cohésion logique, mais les éléments du module présentent en plus des relations dans le temps (par exemple, l'initialisation et la clôture dans un module). [Myers] avoue lui-même que certains modules ne peuvent être réalisés autrement qu'avec une cohésion classique.
La cohésion procédurale est celle où un module réalise plusieurs fonctions et où ces fonctions sont liées par la procédure du problème. Il s'agit en fait des modules représentant un ou plusieurs blocs d'un organigramme de représentation du problème.
La cohésion communicationnelle est la même que la cohésion procédurale, mais les éléments du module communiquent entre eux, c'est-à-dire que soit ils référencent le même ensemble de données, soit ils se passent des données.
Les deux cohésions les plus fortes sont la cohésion fonctionnelle et la cohésion informationnelle.
La cohésion fonctionnelle est la cohésion optimale. Tous les éléments du module tendent à réaliser un objectif unique qui est la fonction. Un module réalise donc une fonction.
Un module peut également réaliser plusieurs fonctions travaillant sur une même structure de données. Dans ce cas, il s'agit de cohésion informationnelle. Ceci revient en fait à un assemblage de modules à cohésion fonctionnelle (avec un "entry point" pour chaque fonction). La cohésion informationnelle permet de garder le secret sur l'information préconisé par [Parnas]. Les changements quant à la structure des données ne devront se faire que dans un seul module.

2.2.2.1.2. Interface.

1° Définition.

L'interface entre 2 modules est l'ensemble d'informations circulant entre ces modules, c'est-à-dire les arguments que le module reçoit et l'ordre dans lequel il les reçoit.
La notion d'interface est très importante car toute la coordination du travail entre les modules va se faire grâce à l'information communiquée entre ces modules.
Il y a 2 moyens de communiquer de l'information entre modules :
- la communication explicite, c'est-à-dire par le biais des arguments appartenant à leur interface;
- la communication implicite, c'est-à-dire par l'entremise de données que les modules se partagent.

Il est en général admis que l'utilisation de la communication explicite exclusivement permet des systèmes plus maîtrisables et plus faciles à maintenir (1). Toute l'information circulant dans un programme modulaire écrit selon la méthode Memo-Proges se fera donc par le canal des seuls interfaces explicites entre modules.
La forme de l'interface d'un module, c'est-à-dire la manière dont il est couplé aux autres, définit le degré de couplage. Comme nous l'avons dit (§ 2.2.1.2.1.), ce couplage, mesure des relations entre modules, doit être minimal afin de garantir une plus grande qualité aux modules.

2° Critère d'évaluation : le couplage.

Tout comme pour la cohésion interne des modules, [Myers] définit plusieurs types de couplages externes qu'il a classifiés selon une échelle de dépendance entre modules.
La dépendance maximale entre 2 modules est appelée le couplage par le contenu. Un module fait une référence

(1) [Myers] fait une critique de la communication implicite (pp.37-39).

directe au contenu d'un autre module (par exemple, change une instruction de l'autre module, référence une variable locale d'un autre module, fait un branchement à une instruction non déclarée comme "entry point", ...).
Le couplage par données communes représente l'utilisation d'un environnement commun, c'est-à-dire que plusieurs modules référencent des structures de données partagées globalement. Le couplage par données externes est identique au couplage par données communes, mais avec des éléments individuels et non des structures de données. Il peut également s'agir d'une référence à une instruction définie externe au module.
Ces 3 couplages présentent de gros désavantages, surtout au niveau de la modifiabilité. Un changement isolé dans un programme est difficile, car il faut inspecter toute la logique de celui-ci. De plus, le fait de référencer des mêmes données rend plus difficile la cohérence du programme. Dès qu'une donnée change, il faut regarder les modules qui sont affectés par ce changement et les recompiler (ou si la compilation est trop coûteuse, on garde des versions différentes des données).
Viennent ensuite 3 types de couplages qui nous intéressent plus particulièrement : le couplage par informations de contrôle, le couplage par structures de données et le couplage par données.
Il y a couplage par informations de contrôle entre des modules, s'ils se passent des informations de contrôle, c'est-à-dire des informations qui ont pour seul objectif de déterminer le comportement du module appelé.
Un couplage par structures de données est défini entre deux modules si ces deux modules sont couplés en référençant la même structure de données, cette structure n'étant pas globale (le nom et la localisation de la structure sont passés par paramètres). Le principal désavantage de ce type de couplage est la transmission de données inutiles pour le module appelé.
Le couplage minimal est le couplage par données, où toutes les données nécessaires en entrée et en sortie du module

appelé sont passées comme arguments. Ces arguments sont des données isolées. L'interface ne contient pas d'informations de contrôle et ne se présente pas comme une structure de données. Ainsi, seules les données nécessaires sont passées par paramètres, c'est-à-dire les données transformées par le module appelé.

2.2.2.2. Transformation du modèle conceptuel des traitements en architecture modulaire.

Pour opérer la transformation du modèle conceptuel des traitements en architecture modulaire, nous avons procédé par une réinterprétation organique des éléments de ce modèle. Cette réinterprétation nous a permis d'établir une classification originale des modules.

Nous avons vu lors de l'analyse conceptuelle, que le modèle conceptuel des traitements pouvait être représenté par la figure 2.4.

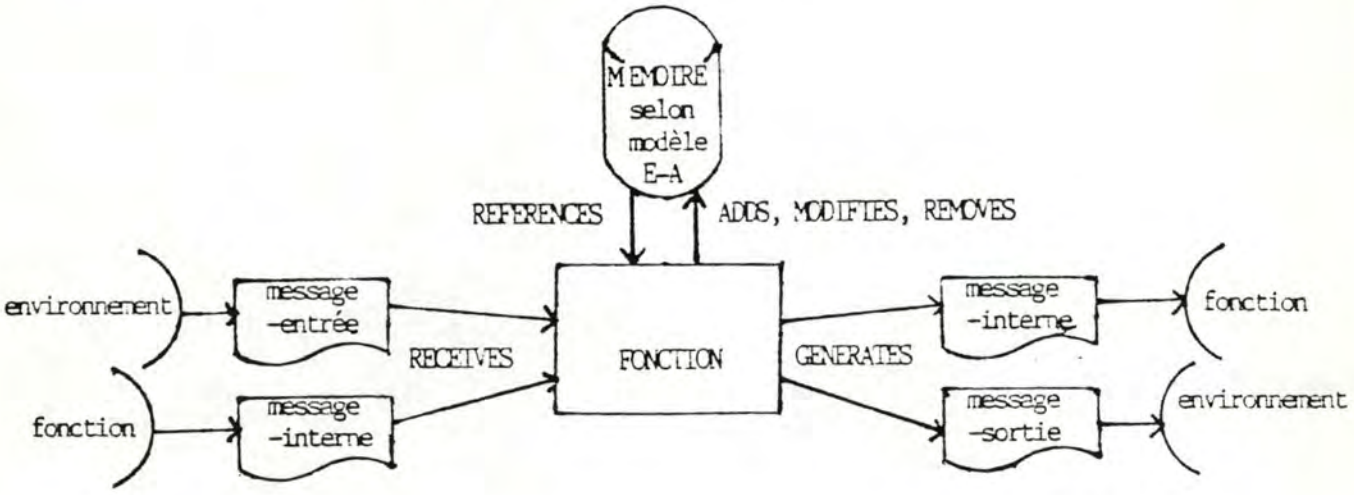


figure 2.4. Modèle conceptuel des traitements.

Ce schéma présente 2 aspects :

- la représentation statique des traitements,
- leur enchaînement dynamique.

1° Transformation du modèle statique des traitements.

La représentation du modèle statique des traitements obtenue à partir de la figure 2.4. est donnée à la figure 2.5., où l'on a supprimé les relations entre les fonctions.

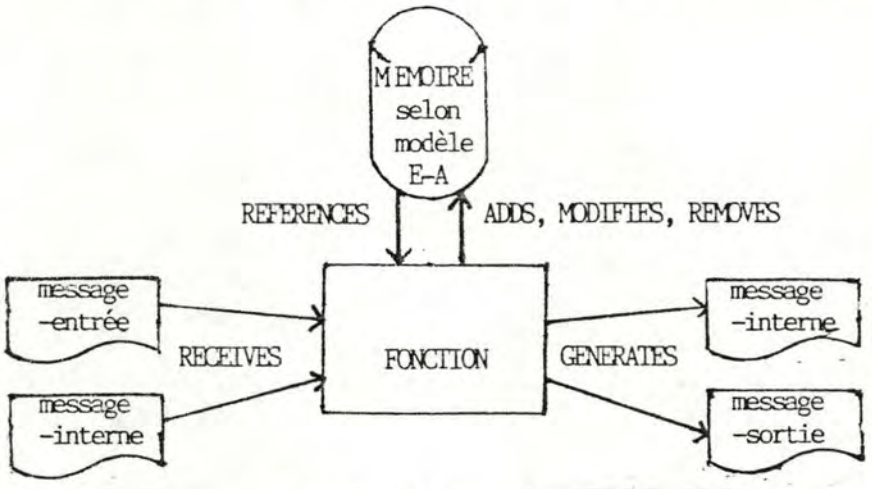


figure 2.5. Modèle statique des traitements.

- L'élément central du modèle est la fonction pour laquelle nous suggérons une réinterprétation sous forme d'un module que nous qualifierons de fonctionnel.
- L'accès aux messages d'entrée et de sortie, ainsi que l'accès aux informations de la mémoire, est réalisé par les modules que nous qualifierons d'entrée-sortie.
  Dans l'architecture logique des données, les informations ont été structurées en termes de fichiers logiques, sur lesquels ont été définies des opérations d'accès (§ 2.2.1.).
  Memo-Proges propose une standardisation des modules d'entrée-sortie sur base :
  - de l'opération à effectuer,
  - du mode d'accès logique,
  - de l'organisation du fichier.

  Les modèles définis dans Memo-Proges, sur base de ces 3 paramètres sont repris dans le tableau de la figure 2.6.

MODELES	OPERATIONS A EFFECTUER	MODE D'ACCES	ORGANISATION
DRANDOMS DRANDOMX	INSERT REPLACE DELETE OBTAIN RECORD (1)	SELECTIF	SEQUENTIELLE INDEXEE
DGROUPS DGROUPX	OBTAIN GROUP	SELECTIF	SEQUENTIELLE INDEXEE
DREAD DREADX	READ	EXHAUSTIF	SEQUENTIELLE INDEXEE
DWRITE	WRITE	EXHAUSTIF	SEQUENTIELLE

figure 2.6. Modèles d'entrée-sortie.

Ces modèles résolvent la plupart des cas, mais il faut respecter certaines conventions de travail. Par exemple, nous utilisons des fichiers et non une base de données (2). De plus, nos fichiers sont monotypes d'enregistrements.

- Les modules d'entrée-sortie doivent être couplés aux modules fonctionnels. Ceci se fait au moyen d'un interface. Nous avons défini l'interface comme l'ensemble des informations circulant entre les modules (§ 2.2.2.1.2.). Entre les modules fonctionnels et les modules d'entrée-sortie, les informations représentent soit les messages d'entrée ou de sortie, soit des informations de la mémoire. L'interface sera représenté par un fichier logique ayant un statut particulier par rapport à chaque module, tel que nous l'avons défini au § 2.2.1.1.. Les modules d'entrée-sortie peuvent être couplés aux modules fonctionnels de 4 manières :
  - les modules READ (RECEIVE) message d'entrée
    et WRITE (GENERATE) message de sortie
    seront couplés au module fonctionnel selon un couplage standard, c'est-à-dire un interface N-M;
  - les modules OBTAIN (REFERENCE) info-mémoire à consulter
    et INSERT (ADD)
    REPLACE (MODIFIE) info-mémoire à mettre à jour
    DELETE (REMOVE)
    seront couplés au module fonctionnel selon un couplage étendu, c'est-à-dire respectivement un interface N-A et U-M.

Chaque type de module d'entrée-sortie réalise un verbe d'accès D.S.L. défini dans le modèle statique.

---

(1) Le langage Cobol impose que toutes les opérations d'accès sélectif à un fichier soient regroupées dans un même module.

(2) On pourrait concevoir des modules réalisant les accès pour une base de données.

En appliquant à la figure 2.6. les règles de transformation que nous venons de proposer, on obtient le schéma Memo-Proges équivalent (1) de la figure 2.7.

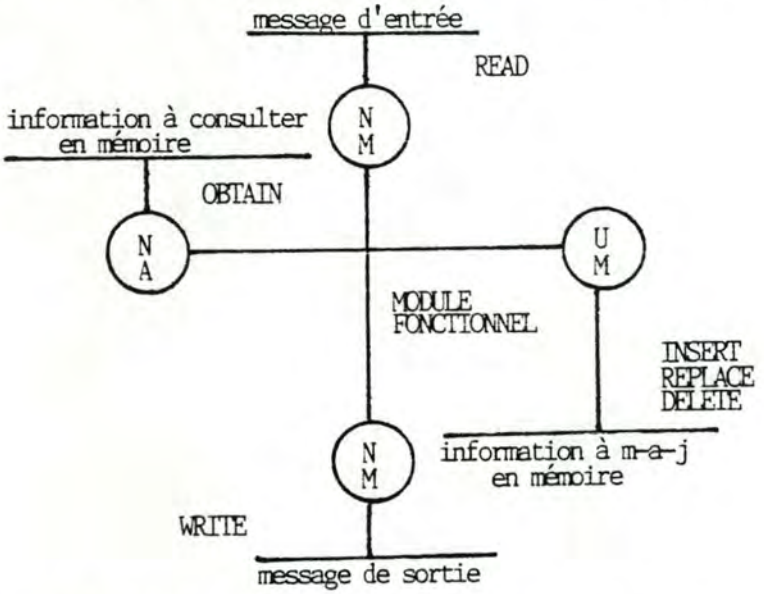


figure 2.7. Couplage des modules d'entrée-sortie aux modules fonctionnels.

2° Transformation du modèle dynamique des traitements.

Une fois les fonctions identifiées, le modèle définit leur enchaînement dynamique. Nous avons vu qu'il existait 2 types d'enchaînements des fonctions :

- soit les fonctions forment un réseau. C'est le cas des problèmes avec flux d'information ;
- soit les fonctions sont groupées dans un catalogue. C'est le cas des problèmes sans flux d'information.

Nous verrons que la manière de réinterpréter l'enchaînement dynamique des fonctions dépend du type de problème.

Dans le cas d'un problème avec flux d'information, [Bodart-Pigneur] a défini 6 schémas dynamiques de base, à l'aide desquels, en les combinant, on peut décrire n'importe quelle dynamique dans un système d'information.

---

(1) La seule différence tient en la représentation. Le schéma Memo-Proges est isomorphe au schéma de l'analyse conceptuelle. La signification des sommets et des arcs est inversée. De plus, sa présentation est verticale.

Nous suggérons de réinterpréter l'enchaînement dynamique des fonctions par des couplages de modules en introduisant éventuellement des modules coordinateurs (1). Dans le cas d'un problème sans flux d'information, les fonctions ne doivent pas être couplées entre elles. On introduira un seul module coordinateur dont le rôle sera de coordonner tous les modules fonctionnels par un système de menus, concrétisation du catalogue des fonctions (2).
Le couplage entre 2 modules, qu'ils soient fonctionnels, ou coordinateurs, est réalisé par un interface représenté par un fichier logique. Ceci peut se schématiser selon la figure 2.8.

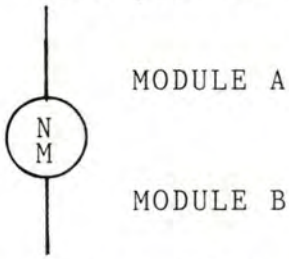


figure 2.8. Couplage des modules fonctionnels et des modules coordinateurs.

Les schémas Memo-Proges représentant le modèle statique et le modèle dynamique des traitements seront combinés afin d'obtenir un schéma général (3) représenté à la figure 2.9.

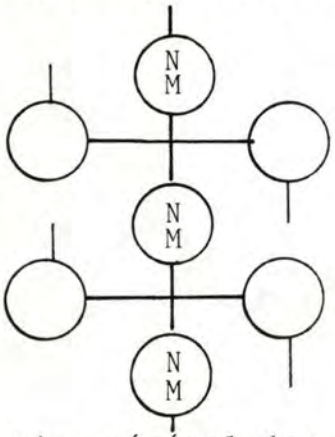


figure 2.9. Schéma général des traitements en Memo-Proges.

---

(1) Nous proposerons dans le chapitre 3 une manière systématique de transformer ces 6 schémas dynamiques de base.

(2) Nous expliquerons dans le chapitre 4 la manière de réaliser ceci.

(3) Le schéma Memo-Proges intègre les 2 aspects statique et dynamique; Contrairement à [Bodart-Pigneur] qui propose 2 schémas indépendants.

2.2.2.3. Spécification des modules et des interfaces.

Spécifier un module revient à spécifier ses attributs :
- spécifier sa fonction, c'est-à-dire son objectif;
- spécifier ses interfaces (en entrée, en sortie, en consultation et mise à jour);
- spécifier sa structure, c'est-à-dire l'algorithme;
- spécifier ses performances.

Pour la spécification des modules fonctionnels, nous reprendrons pour base la spécification des fonctions de l'analyse fonctionnelle. La spécification des modules d'entrée-sortie est prédéfinie puisqu'ils sont standardisés dans Memo-Proges. Quant aux modules coordinateurs, nous présenterons dans les chapitres suivants une spécification prédéfinie pour chaque type de module coordinateur puisque nous avons pu les standardiser.

2.3. REALISATION PHYSIQUE DU SYSTEME.

Pour la réalisation physique du système, nous procéderons de manière tout à fait classique en distinguant la réalisation physique des traitements de la réalisation physique des données.

2.3.1. REALISATION PHYSIQUE DES TRAITEMENTS.

La réalisation physique de l'architecture logique des traitements consistera en son implémentation. Le principal objectif de ce paragraphe sera donc de décrire les éléments de Memo-Proges permettant cette implémentation. Ces éléments seront une concrétisation des éléments de l'architecture logique. Nous donnerons tout d'abord une description des modules et interfaces Memo-Proges, suivie de leur réalisation. Ces deux concepts sont la base de 2 avantages de l'architecture Memo-Proges que nous exposerons ensuite.

2.3.1.1. Module Memo-Proges.

1° Description.

Un module Memo-Proges a la forme d'une sous-routine au sens Fortran du terme.
Tout module doit pouvoir réaliser 3 types d'opérations :

- l'opération ponctuelle est la plus simple. Les appels successifs du module sont mutuellement indépendants. A chaque appel, le module exécute les mêmes opérations.
- l'opération cumulative marque un lien entre les appels successifs : le résultat de l'appel (t-1) intervient comme argument lors de l'appel (t). Il faut donc une initialisation et une terminaison déterminées par l'analyste.
- l'opération itérative concerne un ensemble d'éléments :
  - sur chaque élément de l'ensemble, une opération ponctuelle sera effectuée;
  - sur l'ensemble des éléments, une initialisation et une terminaison d'ordre purement technologiques doivent être effectuées;

  Le module devra être appelé
  - une première fois, afin d'initialiser l'ensemble;
  - une dernière fois, afin de clôturer l'ensemble;
  - pour chaque élément, afin d'effectuer l'opération.

La structure répétitive de l'appel d'un module est la plus générale. Aussi, tout appel de module Memo-Proges est considéré comme une opération itérative, qu'il faudra initialiser et clôturer. Un module cumulatif contient simplement, en plus d'un module itératif, des liens internes dont l'appel ne s'occupe pas. Un module ponctuel aura l'initialisation et la clôture vides.
Tout module Memo-Proges étant considéré comme un module itératif, il faudra qu'il se compose de 3 sections au sens Cobol du terme :

- INIT, section regroupant les opérations à effectuer lorsque le module est appelé pour la première fois;

- TERM, section regroupant les opérations à effectuer lorsque le module est appelé pour la dernière fois;
- CORPUS, section regroupant les opérations à effectuer lorsque le module est appelé pour chaque élément;

La structure d'un module de gestion de fichier aura ainsi la structure suivante :

- Init : ouverture du fichier;
- Term : fermeture du fichier;
- Corpus : opération répétée sur chaque élément du fichier;

2° Réalisation d'un module en Memo-Proges.

Les modules fonctionnels et coordinateurs identifiés lors de l'architecture logique des traitements seront implémentés sous forme de modules Memo-Proges tels que nous venons de les décrire. Quant aux modules d'entrée-sortie, leur implémentation consistera en une actualisation des modèles standards définis à la figure 2.5.
L'implémentation des modules se fera dans le langage M.C.L. (1). Le langage M.C.L. est partiel et intégré en guise d'extension d'un langage hôte. Dans Memo-Proges, le choix du langage hôte s'est porté sur Cobol.

2.3.1.2. Interface Memo-Proges.

1° Description.

L'interface Memo-Proges est un interface banalisé. La banalisation de l'interface a pour objectif principal la simplification maximale des connexions entre modules.
La banalisation de l'interface Memo-Proges se situe à deux niveaux :

- la banalisation statique, c'est-à-dire la définition d'une structure unique d'appel pour tous les modules, quelle que soit leur sémantique. A chaque appel, la forme de l'interface est la même.

---

(1) Le langage M.C.L. (Modular Control Language) fait partie de Memo-Proges et a été développé au groupe S par Clarinval.

- la banalisation dynamique, c'est-à-dire la définition d'une technique unique de contrôle de l'itération. Cela signifie que la méthode de programmation gère (de manière standard) la répétition des appels d'un module. Elle gère, par la même occasion, le contrôle des erreurs (1).

L'interface Memo-Proges a la forme suivante : il est subdivisé en 2 parties : - les données proprement dites, représentées par un fichier virtuel, c'est-à-dire les données passives n'ayant pas pour objectif de déterminer le comportement du programme;
- les informations de contrôle, c'est-à-dire les données actives ayant pour seul objectif de déterminer le comportement du programme.

a. Fichier virtuel.
A chaque appel, on communique au module la structure de données sur laquelle il doit travailler sous forme d'un article-courant d'un fichier. La succession des articles ainsi transmis par les exécutions consécutives d'une même instruction d'appel forme un "fichier virtuel". Le fichier virtuel consiste en fait en une généralisation de la notion de "buffer". C'est un objet dynamique défini dans le temps, contrairement au fichier réel qui est un objet statique défini dans l'espace.

b. Informations de contrôle.
Les informations de contrôle prennent la forme d'indicateurs (2) :
-l'indicateur d'appel %MODE (U-SW1), dont les valeurs conventionnelles indiquent la section (init-corpus-term) à exécuter dans le module appelé (3);
-l'indicateur de retour %STATUS (U-SW2) dont les valeurs conventionnelles signalent au module appelant que le module appelé a rencontré une erreur fatale ou non fatale.

---

(1) La banalisation dynamique est à l'origine de la décentralisation
- de la gestion des erreurs (§ 2.3.1.3.II.ii),
- du contrôle de l'exécution au niveau des modules (§ 2.3.1.3.II.i).

(2) Lorsque la programmation se fait en M.C.L. , l'interface Memo-Proges contient en secret cette partie "informations de contrôle".

(3) %MODE signale aussi quelle opération effectuer pour les modules d'accès sélectif (Insert-Replace-Delete-Obtain).

2° Réalisation d'un interface en Memo-Proges.

Les interfaces identifiés lors de l'architecture logique des traitements sont implémentés sous forme d'interfaces Memo-Proges tel que nous venons de les décrire. Le fichier virtuel concrétise donc le concept de fichier logique.

2.3.1.3. Architecture physique Memo-Proges.

Les concepts de module et d'interface Memo-Proges donnent 2 avantages à l'architecture Memo-Proges :
- une grande souplesse de composition (éclatement et regroupement) et une forte capacité de réutilisation des modules grâce aux 2 qualités de "quasi-autonomie" et "quasi-compatibilité" des modules.
- la coordination du travail entre modules grâce à la décentralisation du contrôle de la dynamique et de la gestion des erreurs.

I. Quasi-autonomie et quasi-compatibilité.

L'architecture physique réalise les couplages entre modules définis par l'architecture logique. Ces couplages sont réalisés au moyen de l'interface fichier virtuel que nous venons de définir. Le fichier virtuel confère 2 qualités aux modules : la quasi-autonomie et la quasi-compatibilité .

La quasi-autonomie signifie qu'un module est capable de fonctionner quasi-seul. Pour ce faire, il faut lui fournir des entrées et exploiter ses sorties. Ceci est réalisé en lui couplant directement

- un module de lecture de ses entrées et
- un module d'écriture de ses résultats

qui permettront de travailler sur des fichiers. C'est le fichier virtuel qui réalise la quasi-autonomie du module. La figure 2.10. présente le schéma de la quasi-autonomie.

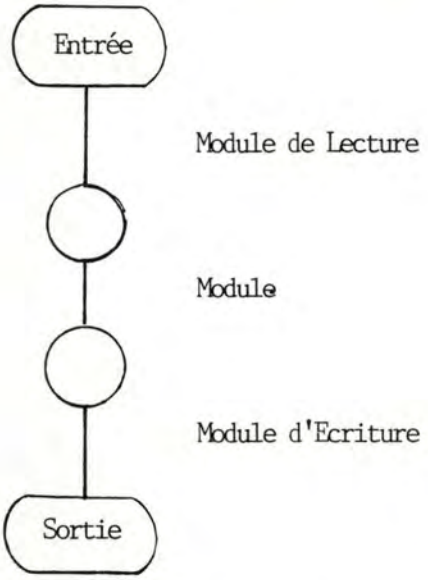


figure 2.5. Quasi-autonomie d'un module.

Le fichier virtuel représente, à chaque pas de l'itération, l'article-courant d'un fichier fourni en mémoire centrale par le module de lecture et que le module d'écriture utilisera. Ceci signifie que les données de l'interface d'un module quelconque peuvent être obtenues ou fournies sur un périphérique sans aucune autre transformation qu'une lecture ou une écriture.

La restriction du préfixe "quasi-" s'explique par le fait que l'autonomie n'est possible que s'il y a couplage d'un module de lecture et d'un module d'écriture.

La quasi-compatibilité revient à pouvoir coupler 2 modules quasi-quelconques. Pour assurer la quasi-compatibilité des modules, il faut que les données de l'interface de tout module puissent être obtenues ou fournies dans la mémoire centrale. C'est aussi le fichier virtuel qui réalise cette quasi-compatibilité des modules.

La restriction du préfixe "quasi-" s'explique par le fait que l'interface des 2 modules à coupler doit représenter le même fichier virtuel.

Ces 2 qualités fondamentales de tout module permettent une grande souplesse de composition (éclatement et regroupement) et une forte capacité de réutilisation des modules. La réutilisation des modules implique en outre l'homogénéité des fonctions qu'ils réalisent.

II. Coordination du travail dans la structure modulaire.

i. Décentralisation du contrôle de la dynamique.

Le principe fondamental du contrôle de la dynamique est la décentralisation. Cette décentralisation se situe à deux niveaux : le niveau local et le niveau global.
Au niveau local, les modules sont dynamiquement complets. En effet, ils contiennent différentes sections (§ 2.3.1.1.) dont l'exécution est entièrement déterminée par l'indicateur de contrôle %MODE contenu dans leur interface (§ 2.3.1.2.).
Au niveau global, tous les modules sont appelés. Le programme se présente comme une cascade d'appels qui se propagent. Il faut donc une racine à la source de tous les appels : c'est le module UROOT. Chaque module doit initialiser et clôturer le module qu'il appelle. Généralement, le programme comportera 3 phases correspondant à 3 appels de UROOT :
- une phase d'initialisation, (appel en mode OPEN) au cours de laquelle tous les modules seront initialisés;
- une phase d'itération, (un appel en mode NEXT);
- une phase de terminaison, (appel en mode CLOSE) au cours de laquelle tous les modules sont clôturés

Le traitement en général ne se faisant pas sur une donnée, mais sur un ensemble de données, cette structure dynamique (singleton) ne suffit pas. La structure itérative de Memo-Proges nécessite la création d'une boucle de traitement. Un module directeur MAIN, appelé par UROOT sera ajouté, et aura pour objectif de créer cette boucle.

Le module UROOT et le module directeur MAIN sont des modules "techniques" ajoutés aux modules "logiques" identifiés lors de l'architecture logique des traitements. Ils peuvent être considérés comme des modules coordinateurs techniques.

ii. Décentralisation de la gestion des erreurs.

Parmi toutes les erreurs susceptibles de se présenter au cours de l'exécution d'un module, nous avons distingué au § 2.3.1.2. 1°b. :
- les ERREURS FATALES,
- les ERREURS NON FATALES,
- par extension, sera considérée comme erreur, l'apparition de l'événement indiquant la FIN normale d'une exploitation d'un programme.

La gestion des erreurs est un problème qui doit être résolu au niveau de toute la structure modulaire. En effet, une erreur a un effet local sur le module où elle se produit, mais elle doit aussi être perçue au niveau global de l'environnement de ce module.
La décentralisation de la gestion des erreurs se fait grâce à l'indicateur %STATUS (§ 2.3.1.2. 1°b.). Grâce à cet indicateur, chaque module pourra savoir si son subordonné a eu une exécution correcte ou pas. Le module appelant peut, s'il y a erreur, soit exécuter une procédure d'exception, soit faire remonter l'indicateur au module dont il est subordonné.
L'erreur peut ainsi remonter jusqu'au module directeur MAIN qui arrêtera l'exécution du programme s'il s'agit d'une erreur fatale ou d'une fin de fichier.

iii. Illustration par un exemple.

Soit le problème suivant :
"A partir d'un fichier des commandes, on voudrait établir les factures à envoyer aux différents clients. A cet effet, on doit consulter le fichier signalétique des clients pour connaître la remise à appliquer à chaque client."
Le problème est résolu par un programme modulaire dont la représentation est donnée à la figure 2.6.

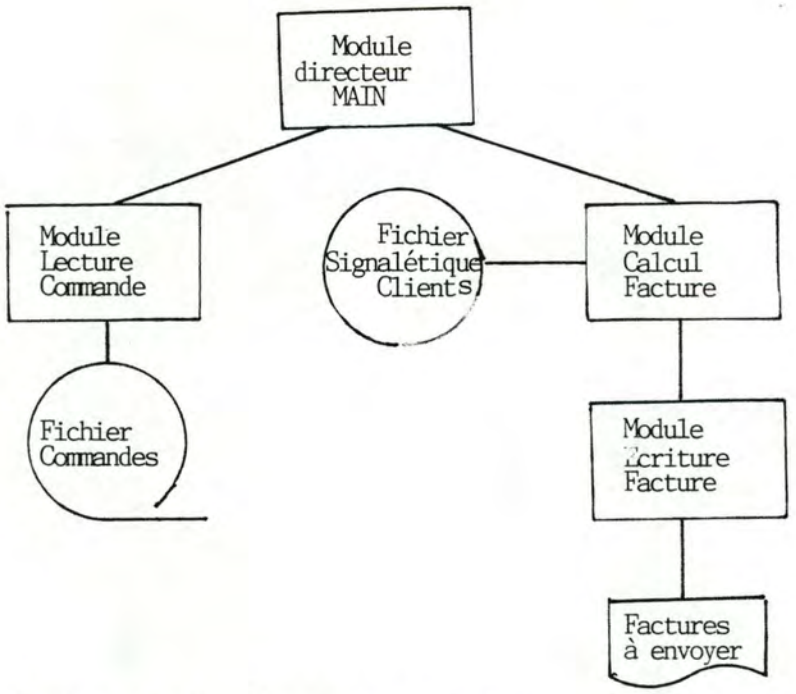
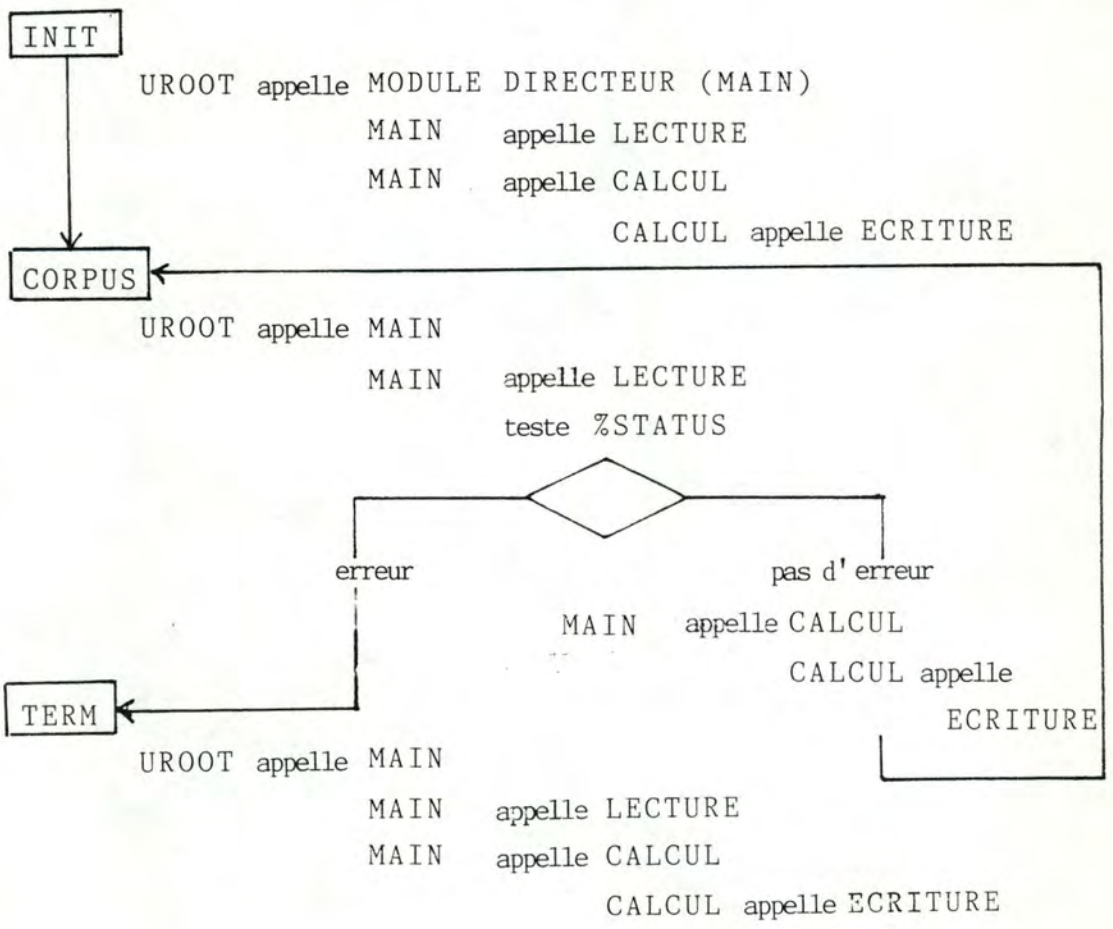


figure 2.6. Programme modulaire de facturation.

Voici le détail de l'exécution de ce programme, afin de montrer comment fonctionne la décentralisation du contrôle de la dynamique.



Parmi toutes les erreurs susceptibles d'apparaître lors de l'exécution d'un module, nous avons vu :

- les erreurs FATALES,
- les erreurs NON FATALES,
  auxquelles sont assimilés les EVENEMENTS SPECIAUX.

Nous allons sur le même exemple montrer comment se fait la décentralisation du contrôle des erreurs.

- Si en cours d'une exécution, il y a ERREUR FATALE (par exemple le fichier des CLIENTS impossible à ouvrir parce que inexistant) : Le module CALCUL FACTURE fera remonter l'erreur jusqu'au module directeur MAIN qui provoquera l'arrêt immédiat de l'exécution.
- S'il y a ERREUR NON FATALE (par exemple, pour une commande traitée, on ne trouve aucun client correspondant) : il y a erreur à signaler et à traiter (ultérieurement), MAIS cette erreur n'empêche pas le traitement normal des commandes suivantes. Le module CALCUL FACTURE traitera l'erreur et l'exécution continuera normalement.
- Par extension, on considèrera comme une erreur l'apparition de l'événement indiquant la fin normale d'une exploitation d'un programme. Comme nous l'avons vu sur l'exemple, l'apparition de l'événement FIN DE FICHIER des COMMANDES sera assimilée à une erreur et le programme se terminera.

2.3.1.4. Evaluation des modules et interfaces Memo-Proges.

Parler de la cohésion interne engendre un paradoxe. En effet, si au lieu de considérer la transformation des "données" par un module tel que le fait Myers , on considère que celui-ci transforme des "fichiers" tel que le fait Clarinval dans Memo-Proges, on est en présence de 2 critères de cohésion opposés. On peut avoir un module qui présente une cohésion fonctionnelle en termes de fichiers, mais une cohésion

faible en termes de données. Inversément, un module peut présenter une cohésion fonctionnelle pour Myers en termes de données, mais une cohésion faible en termes de fichiers. C'est le cas des modules d'accès sélectif de Memo-Proges. Ils définissent une cohésion fonctionnelle en termes de fichiers, puisqu'ils exécutent une seule fonction sur le fichier, une mise à jour. D'autre part, ils définissent une cohésion faible en termes de données, puisqu'ils exécutent plusieurs fonctions telles qu'ajouter un enregistrement, le modifier, le supprimer ...

Par le fait de considérer des transformations de fichiers, les modules Memo-Proges définissent une cohésion classique, caractérisée par le fait que les éléments des modules ont entre eux des relations de temps. Nous avons vu lors de la description des modules Memo-Proges, qu'ils contenaient 3 sections exécutées à des moments différents. Cette structure est due à la considération de fichiers : Init et Term réalisant les opérations d'initialisation et de terminaison relatives au fichier dans son ensemble; Corpus réalisant un traitement sur chaque élément du fichier.

Myers critique cette structure de module, car elle est non-déterministe, c'est-à-dire que le module ne fait pas la même chose à chaque appel :

- premier appel, section Init;
- dernier appel, section Term;
- autres appels, section Corpus;

Pour lui, le comportement d'un module doit être prévisible, c'est-à-dire exécuter des opérations identiques sur des entrées identiques. Autrement dit, le module travaille indépendamment de son environnement. Cette exigence exclut l'utilisation d'un vecteur d'état, c'est-à-dire le passage à l'appel d'informations de contrôle. Un module déterministe correspond en fait à une structure ponctuelle. Or nous venons de voir qu'il existe 2 structures internes plus complexes : la structure cumulative et la structure itérative qui

ne peuvent être ramenées à une structure ponctuelle.
Pour preuve ces 2 exemples :

- un module qui calcule la valeur cumulée, à partir d'entrées successives ne peut être implémenté par une structure ponctuelle, si ce n'est en déclarant globale la variable de cumul, ce que Myers a lui-même fortement critiqué (1).
- un module itératif ne peut réaliser lui-même son initialisation et sa clôture selon le principe de déterminisme. Celles-ci doivent être effectuées par le module appelant, ce qui les rend très dépendants l'un de l'autre. De plus, ceci peut être impossible à implémenter : par exemple, pour la gestion d'un fichier en Cobol (Open Read Close). Myers est conscient de cette limite, mais traite ce problème comme un cas d'exception.

Quant au couplage, Memo-Proges utilise un couplage par structures de données entre ses modules, c'est-à-dire que l'interface entre 2 modules représente une structure de données et non des données isolées. Ce qui pour Myers présente le désavantage de passer des données non nécessaires au module appelé.
Ce qu'il faut considérer quand on recherche l'interface idéal, ce n'est pas un module individuel, mais l'ensemble des modules liés. La simplicité d'un interface entre 2 modules n'entraîne pas nécessairement la simplicité de l'architecture modulaire. Le couplage par données, en effet, rend le module appelé plus simple, puisqu'il ne reçoit que des données utiles. Mais le module appelant dépend du module appelé, car il doit en connaître les besoins. Le module appelant lui-même peut être obligé de traiter des données inutiles pour lui, mais utiles pour un de ses modules subordonnées. L'ensemble des données qui lui sont "nécessaires" peut ainsi être gonflé superficiellement. Tous les modules appelants sont donc très dépendants de leurs subordonnés.

---

(1) Myers pp. 37-39 Common coupling.

Du point de vue des connaissances à avoir sur le module appelé, il est moins difficile de référencer une structure de données complète plutôt que de connaître des données isolées. La définition des structures de données se fera à un niveau global, afin que tous les concepteurs de modules puissent en connaître le contenu.

2.3.2. REALISATION PHYSIQUE DES DONNEES.

L'architecture physique des données met en évidence 2 types de fichiers :

- le fichier virtuel que nous avons défini § 2.3.1.2., qui représente un interface entre les modules ;
- le fichier réel, qui représente un ensemble d'informations existant pour l'utilisateur.

L'intervention de ces fichiers dans l'architecture définit 3 niveaux de modules représentés à la figure 2.11.

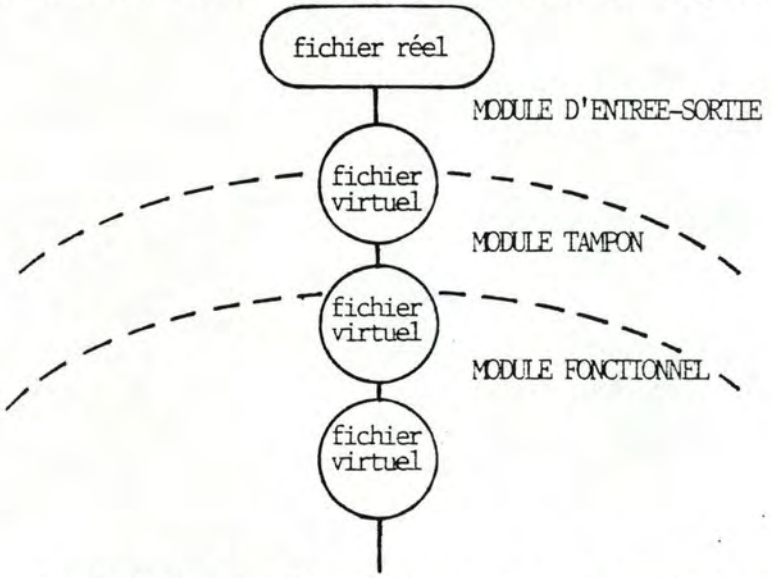


figure 2.11. Représentation des fichiers réels et des fichiers virtuels.

Les modules fonctionnels et coordinateurs établis dans l'architecture logique des traitements. Ces modules n'ont comme interface que des fichiers virtuels.

Les modules d'entrée-sortie interagissent avec l'environnement du programme, soit pour acquérir, soit pour fournir des informations. Ils opèrent la transformation des fichiers réels en fichiers virtuels.

Les modules tampons sont des modules techniques ajoutés aux modules logiques identifiés lors de l'architecture logique des traitements. Ils réalisent les transformations nécessaire lorsqu'un fichier virtuel n'a pas le format d'un fichier réel. Dans une structure de stockage classique, il est tentant de rendre égaux les contenus des fichiers virtuels et fichiers réels pour économiser des modules tampons.

2.4. CONCLUSION.

Nous venons de montrer qu'il est possible d'opérer une réinterprétation organique des modules définis lors de l'analyse conceptuelle à partir de 2 modèles extraits de la méthode Memo-Proges : - le modèle module-fonction et
- le modèle de couplage des modules.

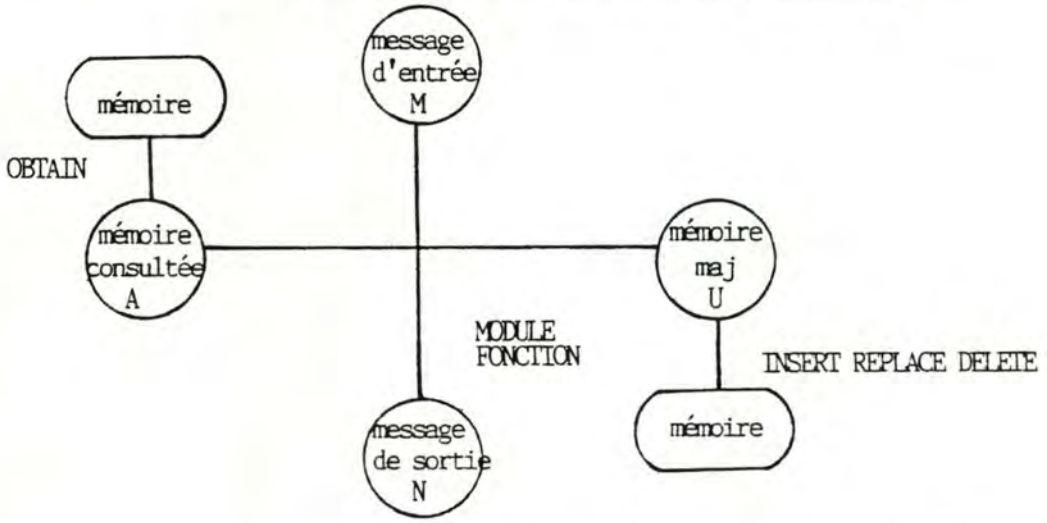


figure 2.12. Modèle module-fonction Memo-Proges.

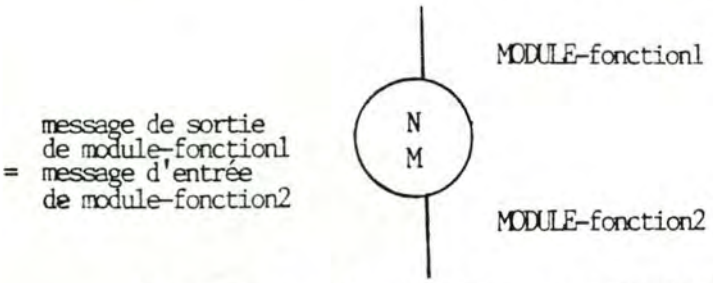


figure 2.13. Modèle de couplage des modules Memo-Proges.

Nous avons ainsi obtenu une architecture modulaire logique dont nous avons expliqué la mise en oeuvre grâce aux outils de programmation Memo-Proges.

CHAPITRE III - DEMARCHE D'ANALYSE D'UN PROBLEME AVEC FLUX D'INFORMATION.

3.0. INTRODUCTION.

Dans ce chapitre, nous proposons une démarche complète d'analyse d'un problème avec flux d'information (§ 1.3.1.1.).
On peut distinguer 2 étapes dans cette démarche :
- l'étape d'analyse conceptuelle, et
- l'étape d'analyse organique.

Ces 2 étapes sont présentées successivement et abondamment illustrées par le cas PETITPAS, cas d'école développé à l'Institut d'Informatique des F.U.N.D.P. de Namur.

3.1. PRESENTATION DU CAS PETITPAS.

La firme PETITPAS S.P.R.L. est une firme de vente par correspondance d'articles d'habillement. Elle se propose d'automatiser la tenue de son stock et le traitement administratif des commandes-clients et des réapprovisionnements-fournisseurs. Dans le cadre de notre travail, nous nous sommes limitées à l'aspect clients (1).

3.2. ANALYSE CONCEPTUELLE : STRUCTURATION DES TRAITEMENTS ET DES DONNEES.

3.2.1. EXPOSE DE LA DEMARCHE D'ANALYSE CONCEPTUELLE.

L'analyse conceptuelle est réalisée par niveaux de traitement (Application, Phase, Fonction). A chaque niveau, on va procéder de la manière suivante :

---

(1) L'énoncé complet du cas PETITPAS est donné dans l'annexe 2.

1°- Spécification du traitement à réaliser;
2°- Idée de solution;
3°- Structuration des informations;
4°- Spécification des informations;
5°- Structuration des traitements,
où l'on distinguera : - la structuration statique, et
- la structuration dynamique.

A propos de cette démarche, deux remarques s'imposent. Premièrement, la phase constituant le repère central de la nomenclature des traitements, la structuration des informations ne se fera qu'à ce niveau. On obtiendra ainsi des sous-schémas conceptuels de données pour chaque phase, que l'on pourra intégrer pour obtenir le schéma conceptuel des données de l'application et que l'on pourra désintégrer pour obtenir les structures d'informations de chaque fonction d'une phase déterminée. Deuxièmement, la fonction constituant le repère élémentaire de la nomenclature des traitements, il n'y aura pas au niveau de celle-ci de structuration des traitements.

3.2.2. APPLICATION DE LA DEMARCHE D'ANALYSE CONCEPTUELLE AU CAS PETITPAS.

3.2.2.1. ANALYSE AU NIVEAU DE L'APPLICATION.

A. Spécification.

Le cas PETITPAS se subdivise en 2 applications :
- application-gestion-client
- application-gestion-fournisseur.

Nous avons uniquement développé l'application-gestion-client. La spécification de cette application se fera en langage D.S.L. de la manière suivante :

```
DEFINE PROCESS Application-gestion-client ;
RECEIVES Bon-commande-client ;
GENERATES Bon-commande-refusé ;
GENERATES Documents-expédition-client ;
DESCRIPTION;
   Objectif.-
     Cette application a pour objectif de gérer automatiquement
     les commandes émanant des clients et par voie de conséquence
     de faire la mise à jour moins du stock des produits.
   Performances.-
     On souhaite :
       - minimiser le nombre de ventes perdues et le nombre
         de livraisons différées par une gestion optimale du
         stock des produits
       - qu'une livraison d'une commande au client ait lieu dans
         les deux jours qui suivent sa réception.
   ;
```

B. Idée de solution.

L'idée de solution suivante a été imaginée en accord avec les services concernés :

" A chaque arrivée du courrier, le service de réception s'assure que les bons de commande provenant des clients sont identifiables et signés. Les bons non signés sont renvoyés au client avec un justificatif de refus. Les bons corrects sont ensuite contrôlés et enregistrés. Une commande est acceptée pour autant qu'au moins une de ses lignes porte sur un produit repris au catalogue ; les commandes incorrectes sont renvoyées au client avec un justificatif de refus.
Les commandes acceptées sont ensuite répercutées dans le fichier des stocks. La partie de la commande livrable sans délai donne lieu à un bon de réquisition ; le reste est mémorisé et donnera lieu à une ou plusieurs livraisons différées.
Lorsque 100 bons de réquisition ont été ainsi émis, on procède à un ordonnancement dont le résultat est un bon de réquisition par série c'est-à-dire un document permettant à un magasinier de préparer sans difficultés les produits relatifs à 100 livraisons en un seul parcours du magasin.
Au terme de ce parcours, les opérateurs d'emballage reconstituent les colis grâce aux bons de réquisition. Si un colis a pu être de la sorte entièrement reconstitué, on déclenche l'impression de la facture et des documents d'expédition. Si un ou plusieurs colis n'ont pu être entièrement reconstitués, le magasinier refait un parcours en magasin à la recherche des produits manquants. Si, malgré ces efforts, un ou plusieurs colis ne peuvent être reconstitués, les corrections adéquates sont apportées au système.
Les commandes complètement traitées seront archivées. "

C. Structuration et spécification des informations.

Le schéma conceptuel des données ne pourra être construit qu'après l'analyse de toutes les phases de l'application, car il résultera de l'intégration des différents sous-schémas conceptuels définis au niveau de chaque phase.

D. Structuration des traitements.

1° Structuration statique : décomposition en phases.

L'idée de solution automatisée développée pour l'application "Gestion-Client" permet de retenir, comme nous l'avons vu au § 1.3.1.2.1.1°, 6 phases automatisables :

- ENREGISTREMENT-DU-BON-DE-COMMANDE-CLIENT
- MAJ-MOINS-DU-STOCK
- ORDONNANCEMENT
- CORRECTION-DU-SYSTEME
- FACTURATION
- ARCHIVAGE.

2°. Structuration dynamique : enchaînement des phases.

L'enchaînement des phases est modélisé au moyen du modèle de la dynamique [Bodart-Pigneur].
L'expression graphique (1) de la logique d'enchaînement des phases est représenté à la figure 3.1.

---

(1) Les conventions graphiques utilisées pour le modèle dynamique ont été exposées au § 1.3.1.2.1.2°, et seront les mêmes pour toutes les représentations selon le modèle dynamique qui suivront.

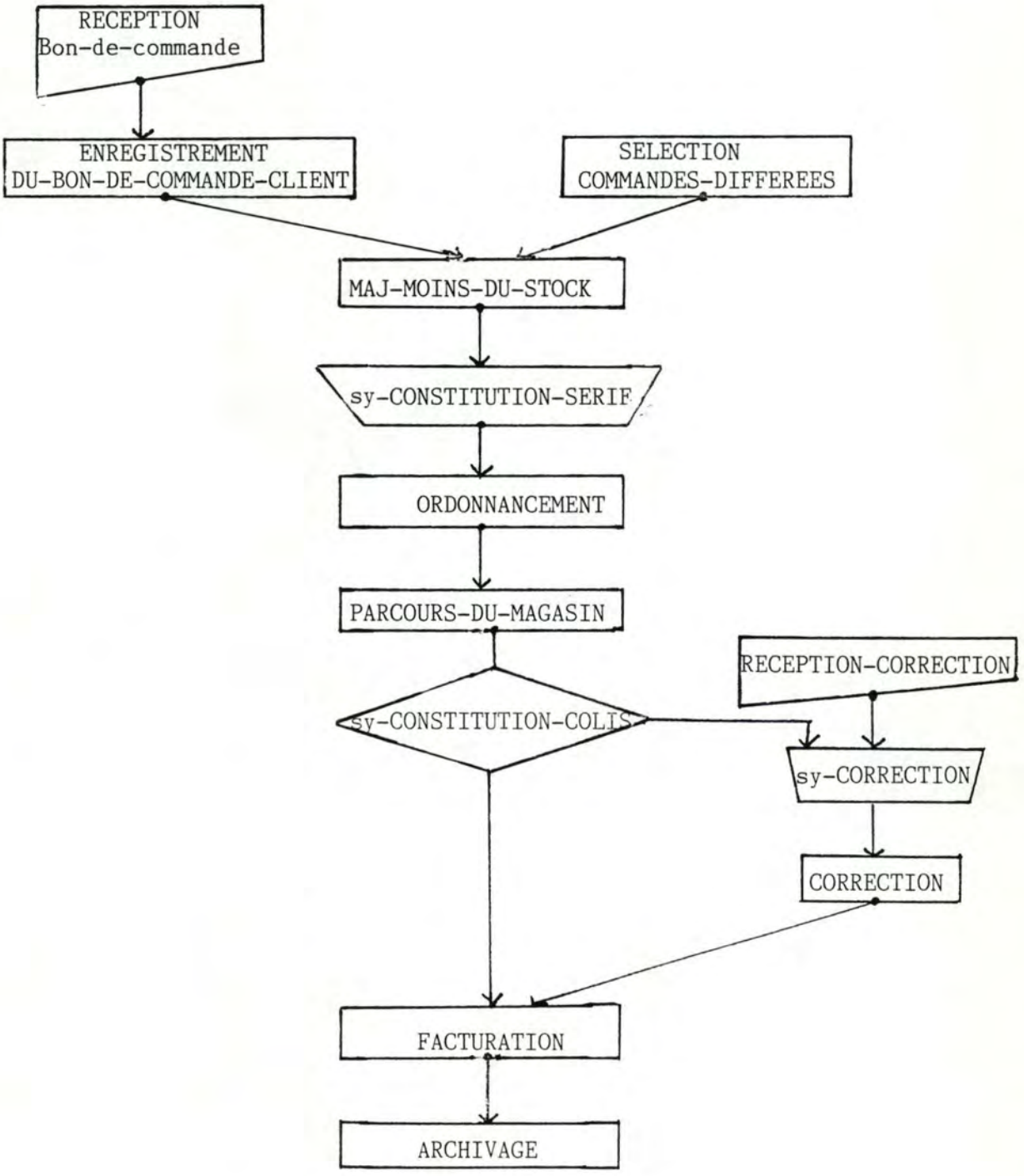


figure 3.1. Modèle dynamique d'enchaînement des phases.

3.2.2.2. ANALYSE AU NIVEAU DES PHASES.

3.2.2.2.1. Phase 1 : Enregistrement du bon de commande client.

A. Spécification.

La spécification de la phase "Enregistrement-du-bon-de-commande-client" s'exprime en langage D.S.L. comme suit :

```
DEFINE PROCESS Phase-enreg-bc-client ;
RECEIVES Bon-commande-signé ;
GENERATES Ligne-commande-acceptée ;
GENERATES Ligne-commande-refusée ;
GENERATES Lettre-d-expédition, nouveau-bon-commande ;
USES Client, produit ;
DESCRIPTION;
  Objectif.-
    Cette phase poursuit 3 objectifs :
        - identifier le client qui a passé commande
        - vérifier si la commande elle-même est correctement
          libellée
        - enregistre dans le système les commandes correctes.
  Performances.-
     On souhaite refuser le moins de bons de commande possible.
```

B. Idée de solution.

Pour atteindre les objectifs assignés à cette phase,

1) On identifie d'abord le client. A cet effet,
   a) on regarde d'abord si l'identité du client (nom + adresse) figure sur le bon de commande. Si elle n'y figure pas, la commande est rejetée ;
   b) si l'identité du client figure sur le bon de commande, on recherche par le numéro ou l'identité du client si ce client figure déjà dans le fichier des clients. S'il s'y trouve, on vérifie la concordance entre les adresses figurant sur le bon de commande et celle reprise dans le fichier ; s'il y a discordance, on modifie l'adresse du fichier. S'il ne s'y trouve pas, on crée un nouveau client.

2) On vérifie ensuite les différentes lignes du bon de commande. A cet effet, pour chaque ligne de commande :
   a) on regarde si le numéro ou le nom du produit apparaît sur la ligne ; si aucun de ces deux éléments n'apparaît, la ligne est marquée ;
   b) si un de ces deux éléments apparaît, on recherche, grâce à lui, si le produit commandé figure dans le fichier des produits ; s'il n'y figure pas, la ligne est marquée.

3) On vérifie enfin le montant total de la commande.
A cet effet, on calcule le montant total à payer par le client et on le compare au total tel qu'il a été calculé par le client. Si l'écart entre ces deux montants excède 10 %, le montant total est marqué.

Si pour une commande, une ligne ou le montant total est marqué, la commande entière est refusée et réexpédiée au client avec une lettre d'explication et un nouveau bon de commande ; sinon, la commande est enregistrée dans le système.

C. Structuration des informations.

De cette idée de solution, on extrait la structure d'informations suivante :

1. MESSAGES
   - bon-de-commande-signé
   - justificatif-refus
   - résultat-contrôle-identité
   - résultat-contrôle-ligne
   - résultat-contrôle-total
2. MEMOIRE ou sous-schéma conceptuel modélisé à l'aide du modèle Entité-Association. L'expression graphique de ce sous-schéma est présentée à la figure 1.4. (1)

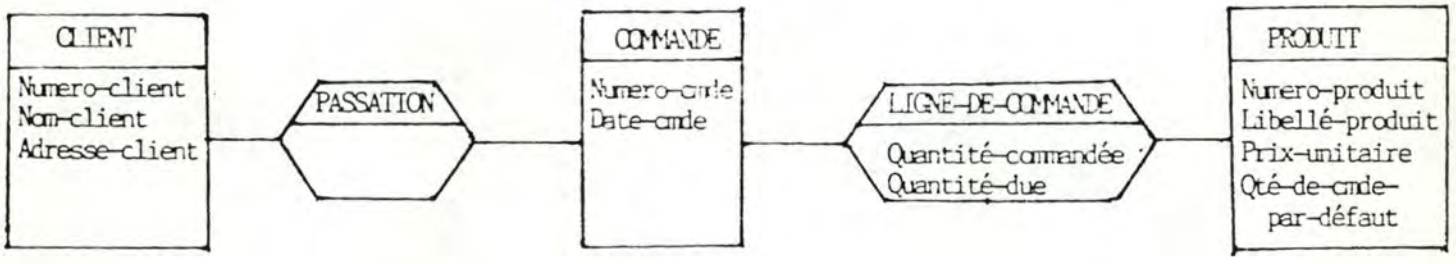


figure 3.2. Sous-schéma conceptuel de la phase "ENREGISTREMENT BON DE COMMANDE CLIENT".

(1) Les rectangles représentent les entités
Les hexagones représentent les associations.

D. Spécification des informations.

La spécification des informations au niveau d'une phase revient à construire ce que l'on appelle communément le dictionnaire des données au niveau de la phase.
Considérant ce point inintéressant pour le travail en cours, nous nous limiterons à spécifier en D.S.L. l'entité Client et l'association Ligne-de-commande-client.

```
DEFINE ENTITY Client;
DESCRIPTION; est considéré comme client, toute personne physique
   ou morale qui a au moins passé à la firme une commande qui
   a permis de l'identifier;
IDENTIFIED BY Numéro-de-client;
RELATED BY Client/commande;
CONSISTS OF Numéro-de-client;
CONSISTS OF Nom-client;
CONSISTS OF Adresse-du-client;
CARDINALITY IS 40 000;

DEFINE RELATION Ligne-de-commande-client;
DESCRIPTION; une occurrence de cette association existe entre
   une commande et un produit quand pour cette commande il
    existe une ligne qui référence le produit;
RELATES commande-client;
        produit;
CONSISTS OF quantité-commandée-client;
CONSISTS OF prix-ligne-cmde-client;
CONSISTS OF quantité-due-client;
```

Les spécifications de tous les messages, entités, associations, attributs de cette phase ainsi que de toutes les autres phases sont données dans le listing [PETITPAS] .
Ce point ne sera plus traité pour les autres phases.

E. Structuration des traitements.

1° Structuration statique : décomposition en fonctions

A partir de l'idée de solution et de la stucture d'informations sous-jacente, on décompose la phase "Enregistrement-du-bon-de-commande-client" en les fonctions suivantes selon les règles définies au § 1.2.2.1. :

AFFICHER <RESULTAT-CONTROLE-IDENTITE >
AFFICHER <RESULTAT-CONTROLE-LIGNE >
AFFICHER <RESULTAT-CONTOLE-TOTAL>
AJOUTER <CLIENT >
MODIFIER <CLIENT >
AJOUTER <LIGNE-COMMANDE >
EDITER <JUSTIFICATIF-DE-REFUS >

2° Structuration dynamique : enchainement des fonctions

L'expression graphique de la logique d'enchaînement des fonctions dans la phase "Enregistrement-du-bon-commande-client" est présenté à la figure 3.3.

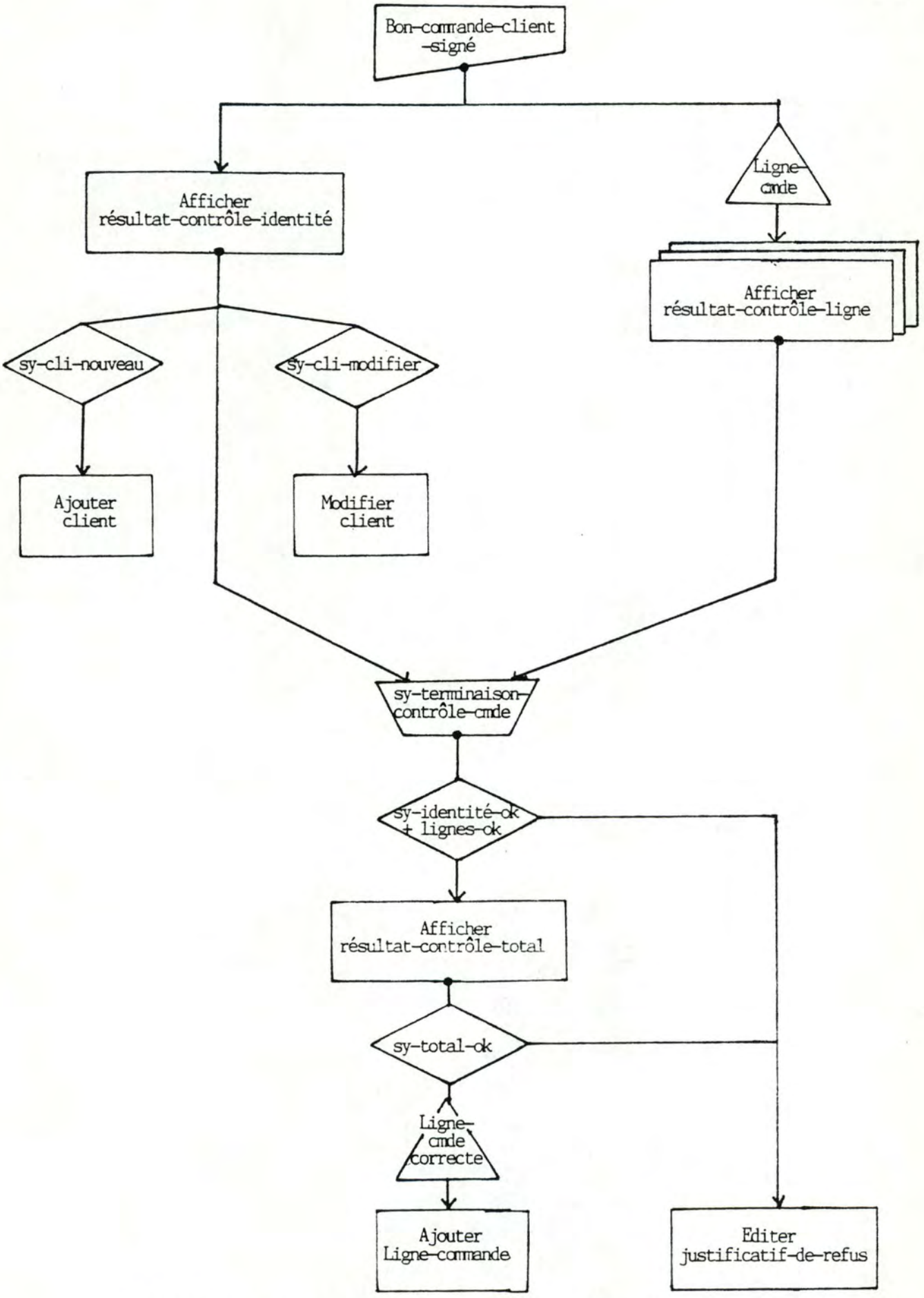


figure 3.3. Dynamique de la phase "ENREGISTREMENT-BON-DE-COMMANDE-CLIENT".

3.2.2.2.2. Phase 2 : Mise à jour moins du stock.

A. Spécification.

La spécification de la phase "Mise-à-jour-moins-du-stock" s'exprime en langage D.S.L. comme suit :

```
DEFINE PROCESS Phase-maj-moins-stock ;
RECEIVES Ligne-commande-acceptée ;
GENERATES Ligne-réquisition ;
USES Produit, ligne-commande-acceptée ;
DESCRIPTION;
    Cette phase met le stock à jour en fonction des produits qui
    sont commandés. Elle crée également les réquisitions corres-
    pondant aux produits commandés;
```

B. Idée de solution.

La mise à jour des stocks et la création des réquisitions, objectifs de la phase 2 , obéissent aux règles suivantes pour chaque ligne d'une commande enregistrée :

CONDITIONS	1	2	3
Stock épuisé et quantité-en-stock = 0	Y		
Quantité-en-stock $\geq$ Quantité-cmdée-client		Y	
Quantité-en-stock $<$ Quantité-cmdée-client			Y
ACTIONS SUR PRODUIT			
(nouv)Quantité-en-stock = (anc)Quantité-en-stock - Qté-réquisitionnée-client		*	
Quantité-en-stock = 0			*
Quantité-différée = (anc)Quantité-différée + Quantité-cmdée-client - Qté-réquisitionnée-client			*
CREER UNE LIGNE-REQUISITION-CLIENT POUR LAQUELLE :			
Qté-réquisitionnée-client = Quantité-en-stock			*
Qté-réquisitionnée-client = Quantité-cmdée-client		*	
ACTION SUR LIGNE-COMMANDE-CLIENT			
Quantité-due-client = Quantité-cmdée-client - Qté-réquisitionnée-client		*	*

C. Structuration des informations et spécification.

De cette idée de solution, on extrait la structure d'informations suivante :

1. MESSAGES :

   ligne-commande-acceptée
   ligne-réquisition

2. MEMOIRE :

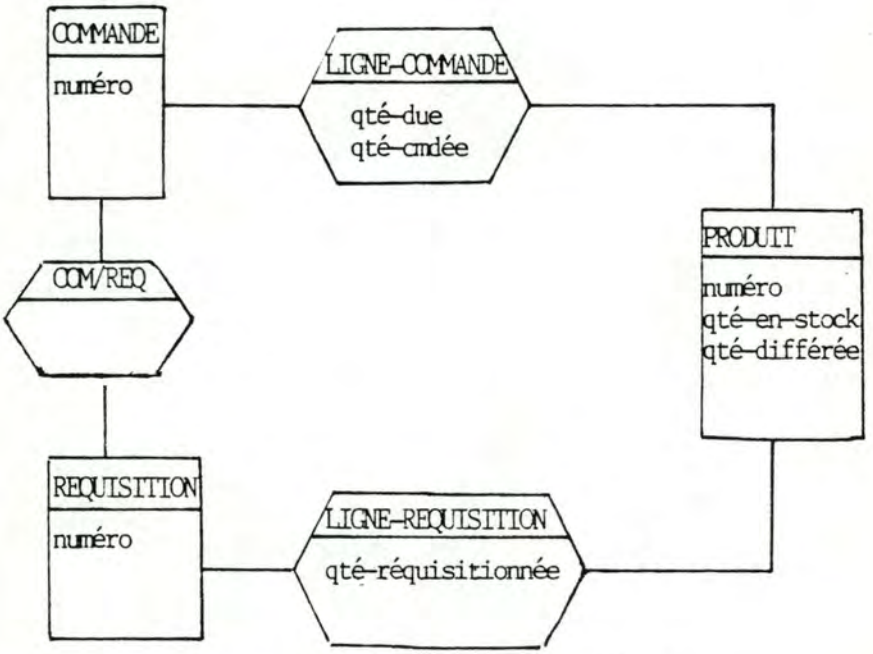


figure 3.4. Sous-schéma conceptuel de la phase MAJ-MOINS-DU-STOCK.

On peut trouver la spécification de ces informations dans le listing PETITPAS .

D. Structuration des traitements.

1° Structuration statique : décomposition en fonctions.

A partir de l'idée de solution et de la structure d'informations sous-jacente, on décompose cette phase en les fonctions suivantes selon les règles définies au § 1.2.2.1. :

AFFICHER <RESULTAT-CONTROLE-LIGNE-COMMANDE>
MODIFIER <PRODUIT>
AJOUTER <LIGNE-REQUISITION>
MODIFIER <LIGNE-COMMANDE>

2° Structuration dynamique : enchaînement des fonctions.

L'expression graphique de la logique d'enchaînement des fonctions dans cette phase est la suivante :

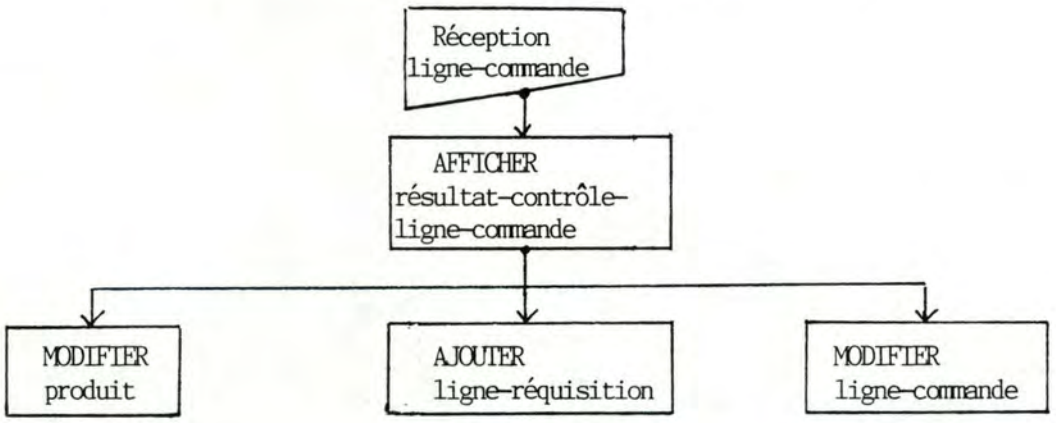


figure 3.5. Modèle dynamique de la phase MAJ-MOINS-DU-STOCK.

3.2.2.2.3. Phase 3 : Ordonnancement.

A. Spécification.

La spécification de la phase "Ordonnacement" s'exprime en langage D.S.L. comme suit :

```
DEFINE PROCESS Phése-ordonnancement;
   RECEIVES Ligne-réquisition;
   GENERATES Ligne-expédition;
   GENERATES Bon-par-casier, Bon-par-série;
   USES Produit;
   DESCRIPTION;
      cette phase se décompose en 2 parties :
      - regrouper N <= 100 commandes,
      - ordonnancer ce lot de N commandes, de façon à
        optimiser le parcours en magasin;
```

B. Idée de solution.

L'ordonnancement, objectif de la phase 3, va se faire comme suit :

- attribuer un numéro de lot à chaque série (au maximum 100 commandes dans une série),
- créer une entité expédition pour chaque commande de la série selon les règles suivantes :

Il existe au moins une réquisition pour cette ligne Quantité-due-client > 0 Stock épuisé	Y Y	Y N	N N	N Y Y	N Y N
Créer une Ligne-expédition-client pour laquelle Quantité-expédiée -client = Somme-quantités-réquisitionnées	*	*			
Créer une ligne-expédition-client pour laquelle Quantité-expédiée-client = 0				*	*
EPUISE → motif-du-refus-client				*	
RESTE RECE PLUS TARD → motif-du-refus-client	*				
A RECEVOIR ULTERIEUREMENT → motif-du-refus-client					*
Quantité-due-client = 0			*		

- imprimer 2 bordereaux :

BON PAR SERIE :

LOT N° :	CASIER 1	CASIER 2	. . .	CASIER 10
N° PRODUIT LBELLE	QTE	QTE		QTE
....	....	....		....
....	....	....		....

(les produits sont rangés dans l'ordre de leur prélèvements.)

BON PAR CASIER

LOT N° :	CASIER N° : ..
N° COMMANDE	annotation de réquisition
.....	.....
.....	.....

C. Structuration des informations et spécifications.

De cette idée de solution, on extrait la structure d'informations suivante :

1.MESSAGES :

Ligne-réquisition
Ligne-expédition
Bon-par-casier
Bon-par-série

2. MEMOIRE :

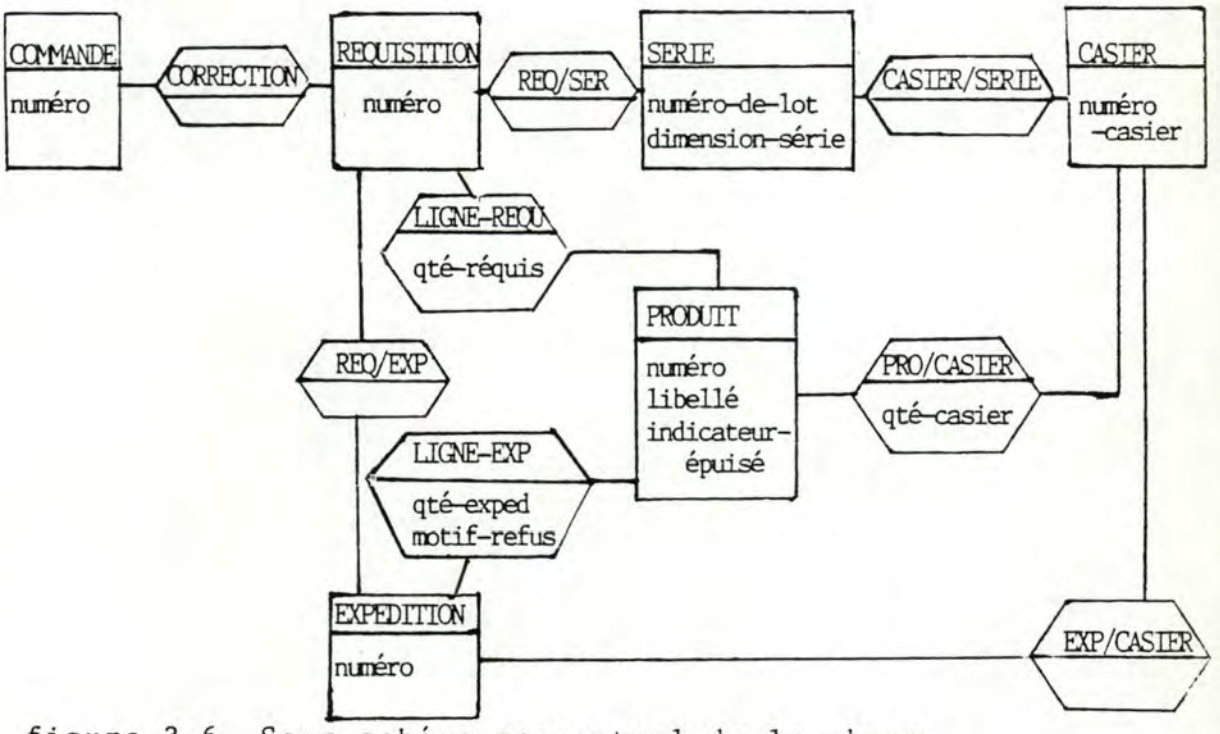


figure 3.6. Sous-schéma conceptuel de la phase ORDONNANCEMENT.

On peut trouver la spécification de ces informations dans le listing PETITPAS .

D. Structuration des traitements.

1° Structuration statique : décomposition en fonctions.

A partir de l'idée de solution et de la structure d'informations sous-jacente, on décompose cette phase en les fonctions suivantes selon les règles définies au § 1.2.2.1. :

AFFICHER <RESULTAT-CONTROLE-LIGNE-REQUISITION>
AJOUTER < LIGNE-EXPEDITION >
EDITER < BON-PAR-CASIER >
EDITER < BON-PAR-SERIE >

2° Structuration dynamique : enchaînement des fonctions.

L'expression graphique de la logique d'enchaînement des fonctions dans cette phase est la suivante :

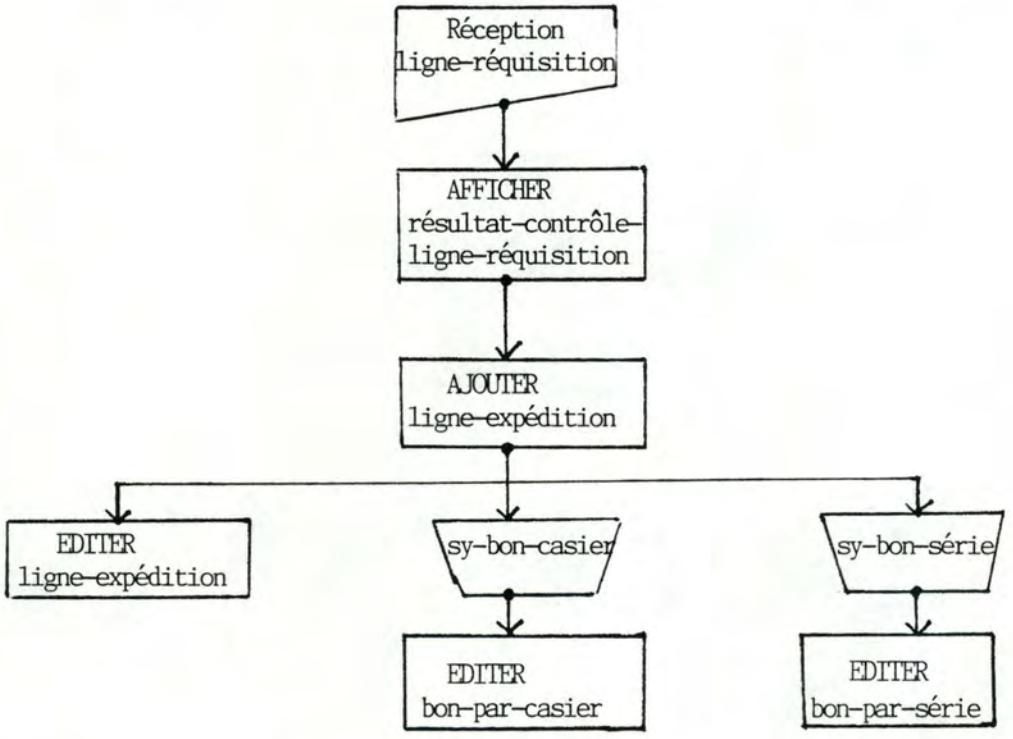


figure 3.7. Modèle dynamique de la phase ORDONNANCEMENT.

3.2.2.2.4. Phase 4 : Correction du système.

A. Spécification.

La spécification de la phase "Correction-du-système" s'exprime en langage D.S.L. comme suit :

```
DEFINE PROCESS Phase-correction-du-système;
   RECEIVES Correction;
   USES Ligne-expédition;
   USES Produit;
   USES Ligne-commande-acceptée;
   DESCRIPTION;
      mettre à jour le stock, le contenu de l'expédition
      et les contenus des quantités-dues afin de les
      adapter à l'état réel du stock;
```

B. Idée de solution.

La phase 4, correction, obéit aux règles suivantes :

pour chaque correction,

ACTION SUR PRODUIT : (nouv)qté-en-stock =
(anc)qté-en-stock - qté-manquante

ACTION SUR LIGNE-EXPEDITION : (nouv)qté-expédiée =
(anc)qté-expédiée - qté-manquante

ACTION SUR LIGNE-COMMANDE : (nouv)qté-due =
(anc)qté-due + qté-manquante

C. Structuration des informations et spécification.

De cette idée de solution, on extrait la structure d'infomations suivantes :

1. MESSAGES :

correction

2. MEMOIRE :

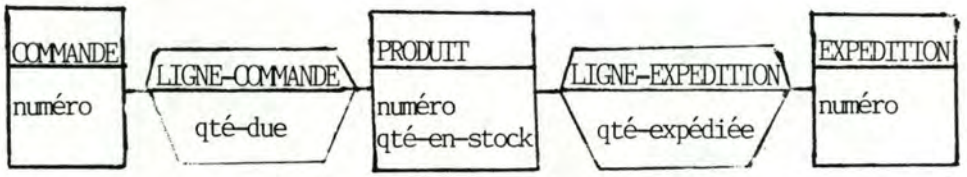


figure 3.8. Sous-schéma conceptuel de la phase CORRECTION-DU-SYSTEME.

On peut trouver la spécification de ces informations dans le listing PETITPAS .

D. Structuration des traitements.

1° Structuration statique : décomposition en fonctions.

A partir de l'idée de solution et de la structure d'informations sous-jacente, on décompose cette phase en les fonctions suivantes selon les règles définies au § 1.2.2.1. :

MODIFIER <LIGNE-EXPEDITION>
MODIFIER <LIGNE-COMMANDE>
MODIFIER <PRODUIT>

2° Structuration dynamique : enchaînement des fonctions.

L'expression graphique de la logique d'enchaînement des fonctions dans cette phase est la suivante :

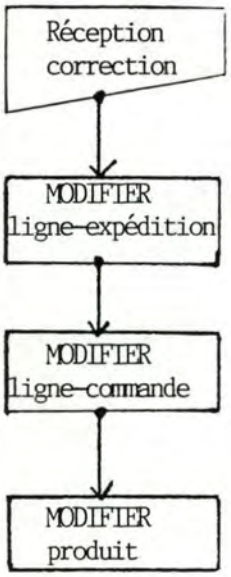


figure 3.9. Modèle dynamique de la phase CORRECTION-DU-SYSTEME.

3.2.2.2.5. Phase 5 : Facturation.

A. Spécification.

La spécification de la phase "Facturation" s'exprime en langage D.S.L. comme suit :

```
DEFINE PROCESS Phase-facturation;
   RECEIVES Ligne-expédition;
   GENERATES Bon-livraison-client;
   GENERATES Etiquette-adress;
   GENERATES Ligne-bordereau-transport;
   USES Ligne-commande-acceptée;
   USES Client;
   DESCRIPTION;
      pour une expédition donnée, construire et imprimer le
      bon de livraison du client et ses documents
      d'accompagnement;
```

B. Idée de solution.

La phase 5, facturation, consiste en :

- imprimer un bon de livraison selon les règles suivantes :

  1. Générer l'en-tête du bon de livraison avec

     numéro-bon-livraison-client → numéro
     date-du-jour → date
     date-commande-client → réf de commande
     numéro-commande-client → notre référence
     nom-client → réf du client
     prénom-client
     adresse-client

  2. Pour chaque ligne de l'expédition
     - garnir une ligne du bon de livraison avec
       numéro-produit → numéro de produit
       libellé-produit → libellé du produit
       quantité-expédiée-client → quantité livrée
       (cette zone est mise à blanc si la valeur à imprimer = 0)
       prix-unitaire → prix unitaire
       poids-unitaire * quantité-livrée → poids du colis
       motif-du-refus-client → motif de non livraison
     - totaliser
       somme-poids-unitaires * quantité-livrée → poids du colis
       somme des montants → total montant ligne
     - garnir la clôture du bon de livraison avec
       frais d'expédition = 40 si c'est une première expédition et s'il existe au moins une quantité-expédiée-client > 0
       = 0 sinon
       total montant ligne + frais d'expédition → total-à-payer

- imprimer une étiquette adresse au nom du client
- générer une ligne-bordereau-transport si le poids du colis est positif

C. Structuration des informations et spécification.

De cette idée de solution, on extrait la structure d'informations suivante :

1. MESSAGES :

   bon-de-livraison-client
   étiquette-adresse
   ligne-bordereau-transport

2. MEMOIRE :

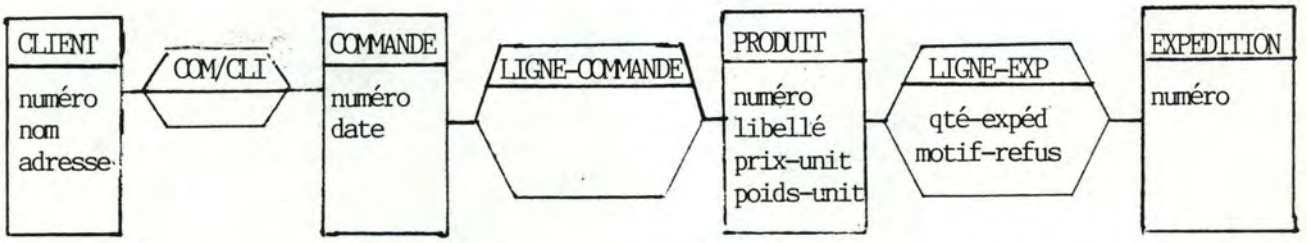


figure 3.10. Sous-schéma conceptuel de la phase FACTURATION.

On peut trouver la spécification de ces informations dans le listing [PETITPAS].

D. Structuration des traitements.

1° Structuration statique : décomposition en fonctions.

A partir de l'idée de solution et de la structure d'informations sous-jacente, on peut décomposer cette phase en les fonctions suivantes selon les règles définies au § 1.2.2.1. :

EDITER <BON-LIVRAISON-CLIENT>
EDITER <ETIQUETTE-ADRESSE >
EDITER <LIGNE-BORDEREAU-TRANSPORT>

2° Structuration dynamique : enchaînement des fonctions.

L'expression graphique de la logique d'enchaînement des fonctions dans cette phase est la suivante :

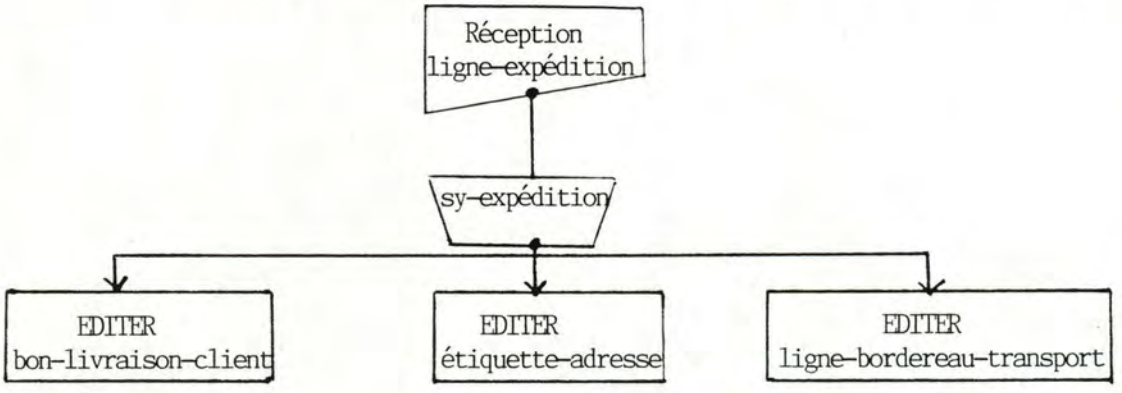


figure 3.11. Modèle dynamique de la phase FACTURATION.

3.2.2.2.6. Phase 6 : Archivage.

A. Spécification.

La spécification de la phase "Archivage" s'exprime en langage D.S.L. comme suit :

```
DEFINE PROCESS Phase-archivage;
   RECEIVES ligne-commande-acceptée;
   USES ligne-commande-différée;
   USES ligne-commande-archivée;
   DESCRIPTION;
        cette phase fait l'archivage des commandes traitées
        - soit vers les commandes archivées si elles sont
          totalement traitées,
        - soit vers les commandes différées sinon;
```

B. Idée de solution.

La phase 6, archivage, obéit à la règle suivante :

```
si toutes les lignes de la commande traitée
ont leur qté-due = 0
     alors la commande est archivée,
     sinon elle est différée.
```

C. Structuration des informations et spécification.

De cette idée de solution, on extrait la structure d'informations suivante :

1. MESSAGES :

   ligne-commande-acceptée
   ligne-commande-archivée
   ligne-commande-différée

2. MEMOIRE :

figure 3.12. Sous-schéma conceptuel de la phase ARCHIVAGE.

On peut trouver la spécification de ces informations dans le listing [PETITPAS].

D. Structuration des traitements.

1° Structuration statique : décomposition en fonctions.

A partir de l'idée de solution et de la structure d'informations sous-jacente, on décompose cette phase en les fonctions suivantes selon les règles définies au § 1.2.2.1. :

AFFICHER <RESULTAT-CONTROLE-LIGNE-COMMANDE>
ARCHIVER <LIGNE-COMMANDE>
AJOUTER <LIGNE-DIFFEREE>
SUPPRIMER <LIGNE-DIFFEREE>

2° Structuration dynamique : enchaînement des fonctions.

L'expression graphique de la logique d'enchaînement des fonctions dans cette phase est la suivante :

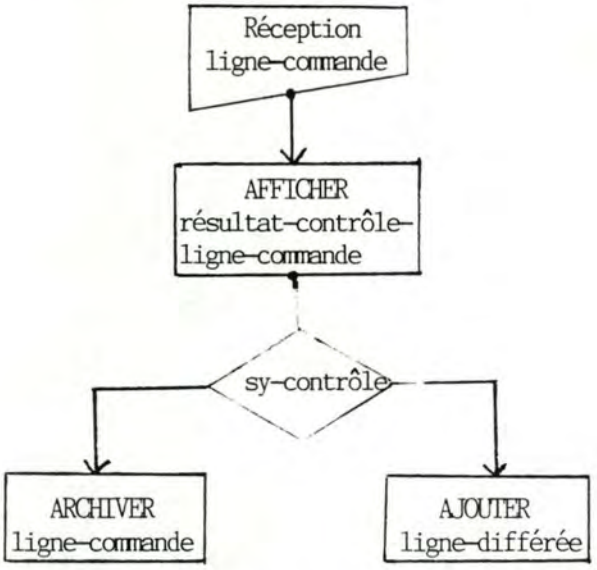


figure 3.13. Modèle dynamique de la phase ARCHIVAGE.

3.2.2.3. ANALYSE AU NIVEAU DES FONCTIONS.

Nous nous limiterons à spécifier les fonctions en langage D.S.L. de la phase 1 :

AFFICHER <RESULTAT-CONTROLE-IDENTITE>
AFFICHER <RESULTAT-CONTROLE-LIGNE>
AFFICHER <RESULTAT-CONTROLE-TOTAL>
AJOUTER <CLIENT>
MODIFIER <CLIENT>
AJOUTER <LIGNE-COMMANDE>
EDITER <JUSTIFICATIF-DE-REFUS>

A. Spécification des fonctions.

DEFINE PROCESS Afficher-résultat-contrôle-identité;
RECEIVES bon-commande-client-signé;
GENERATES Cmde-identité-marquée;
USES numéro-client, nom-client, adresse-client;
DESCRIPTION;

n°-client existe sur B.C.	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
nom-client existe sur B.C.	N	Y	Y	Y	N	N	Y	Y	Y	N	N	N	Y	Y	Y
adresse existe sur B.C.		N	Y	Y	N	N	N	N	N	Y	Y	Y	Y	Y	Y
n-cli (B.C.)~ n°-clients				Y	N	Y	Y	N	Y	Y	N	Y	Y	Y	N
nom-cli (B.C.) = nom (n-cli (B.C.))						Y	N					Y	Y	N	
nom-cli, adresse (B.C.) = nom, adresse			Y	N											Y
adresse (B.C.) = adresse (n-cli (B.C.))									Y	N		Y	N		
marque = 1 (numero-client à ajouter)													*		
marque = 2 (modifier adresse-client)			*											*	*
marque = 3 (corriger B.C.)		*	*												*
marque = 4 (refuser commande)	*	*		*		*	*		*	*					

description des conditions :

n°-client existe B.C. — Existe-t-il un numéro de client sur le bon de cmde ?

nom-client existe sur B.C. — Existe-t-il un nom et un prénom sur le bon de cmde ?

adresse existe sur B.C. — Existe-t-il une adresse complète sur le bon de cmde?

n-cli (B.C.)~ n°-clients — Existe-t-il dans la collection des clients, un client dont le numéro est le même que celui inscrit sur le bon de cmde ?

nom-cli (B.C.) = nom (n-cli (B.C.)) — Les noms et prénoms du bon de commande sont-ils les mêmes que ceux trouvés dans la collection des clients pour le client portant le numéro du bon de cmde ?

nom-cli, adresse (B.C.) = nom, adresse	Existe-t-il dans la collection des clients, un client dont les nom, prénom et adresse complète sont les mêmes que ceux du bon de cmde ?
adresse (B.C.) = adresse (n-cli (B.C.))	L'adresse du bon de cmde est-elle la même que celle trouvée dans la collection des clients pour le client portant le numéro du bon de cmde ?

DEFINE PROCESS Afficher-résultat-contrôle-ligne;
RECEIVES cmde-ligne;
GENERATES cmde-ligne-marquée;
USES numéro-produit, libellé-produit, qté-de-cmde-par-défaut;
DESCRIPTION;

n°-produit existe sur B.C.	N	Y	Y	Y	N	N	N	Y	Y	Y	Y	Y	Y	Y
libellé existe sur B.C.	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
n-produit (B.C.)~ n°-produits		N	Y	Y				Y	Y	Y	Y	Y	N	N
libellé (B.C.) = libellé (N-produit)								Y	Y	N	N	N		
libellé (B.C.)~ libellés					N	Y	Y			Y	Y	N	N	Y
qté-cmdée = M * qté-de-cmde-par-défaut		Y	N		Y	N	Y	N	Y	N			Y	N
marque = 1 (ajuster qté-cmdée)			*					*						
marque = 2 (ajouter num-produit)				*									*	
marque = 3 (changer num-produit)									*					
marque = 4 (abandonner ligne)	*	*		*								*	*	
marque = 5 (ajuster qté-cmdée et ajouter num-produit)							*							
marque = 6 (ajuster qté-cmdée et changer num-produit)											*			*
marque = 0 (accepter ligne)			*				*							

DEFINE PROCESS Afficher-résultat-contrôle-total;
RECEIVES cmde-identité-ligne-ok;
GENERATES cmde-total-marqué;
DESCRIPTION;
si la différence entre le total à payer inscrit sur le bon de cmde et celui calculé par l'ordinateur excède 10%, alors la marque-total = 1 (total refusé)
sinon la marque-total = 0 (total accepté)

DEFINE PROCESS Ajouter-client;
RECEIVES cmde-ajoût-cli;
ADDS client;
GENERATES cmde-identité-ok;
DESCRIPTION;
attribuer un numéro au nouveau client et l'ajouter à la collection des client;

```
DEFINE PROCESS Modifier-client;
   RECEIVES cmde-modif-cli;
   MODIFIES client;
   GENERATES cmde-identité-ok;
   DESCRIPTION;
      modifier l'adresse du client dans la collection des clients;


DEFINE PROCESS Ajouter-ligne-commande;
   RECEIVES ligcmd-enreg;
   ADDS ligne-commande;


DEFINE PROCESS Editer-justificatif-refus;
   RECEIVES cmde-refusée;
   GENERATES justificatif;
   DESCRIPTION;
      éditer le justificatif de refus à destination du client;
```

B. Idée de solution.

L'idée de solution d'une fonction est donnée dans la partie "DESCRIPTION" de sa spécification.

C. Structuration et spécification des informations.

Comme nous l'avons vu au § 3.2.1., la structuration des informations d'une fonction s'obtient par désintégration du sous-schéma conceptuel de la phase à laquelle elle appartient. Quant à la spécification de ces informations, elle a été réalisée au niveau de la phase.

3.3. ANALYSE ORGANIQUE :

IDENTIFICATION ET SPECIFICATION DES COMPOSANTS.

3.3.1. EXPOSE DE LA DEMARCHE D'ANALYSE ORGANIQUE.

Au terme de l'analyse conceptuelle, nous avons obtenu deux modèles :
- le modèle des informations et
- le modèle des traitements.

Ces deux modèles nous serviront de base pour construire dans un premier temps l'architecture des données et dans un second temps l'architecture des traitements dans le cadre de l'analyse organique.

3.3.2. APPLICATION DE LA DEMARCHE D'ANALYSE ORGANIQUE AU CAS PETITPAS.

3.3.2.1. Architecture des données.

Au niveau conceptuel, le modèle des informations a été réalisé par phase parce qu'elle représente l'instant informatique de perception pour l'utilisateur. C'est en effet à la phase qu'il fournit ses données, et c'est de la phase qu'il attend des résultats.

1° Transformation des informations en fichiers logiques.

L'ensemble des messages d'entrée (RECEIVED) et l'ensemble des messages de sortie (GENERATED) deviennent des fichiers logiques respectivement de statuts M et N. Il n'existe pas en Memo-Proges de règles de construction des fichiers logiques quant au contenu de leurs articles. Nous suggérons, au niveau du fichier logique, de faire correspondre à chaque message, un article de ce fichier (1).

(1) Nous avons adopté cette règle dans un souci de généralité. Nous pensons, en effet, que c'est au niveau du fichier physique que les règles de construction devront se différencier en fonction des modes de stockage des données et du mode d'exploitation utilisés.

Quant à la mémoire consultée (REFERENCED) ou mise à jour (ADDED, MODIFIED, REMOVED), elle se transforme en fichiers logiques respectivement de statuts A et U. Nous suggérons qu'à toute entité ou association consultée ou mise à jour corresponde un article du fichier logique.
Cette transformation peut être schématisée par la figure 3.14..

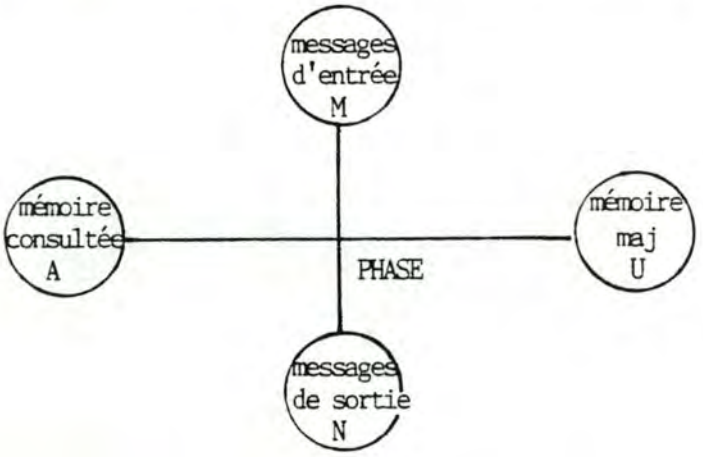


figure 3.14. Fichiers logiques pour une phase.

2° Transformation des fichiers logiques en fichiers physiques.

Le premier type de fichier physique est le fichier réel qui permettra la communication des informations entre l'utilisateur et la phase.
Les fichiers logiques représentant les messages d'entrée et de sortie seront transformés en fichiers réels respectivement de données et de résultats pour l'utilisateur. Nous n'avons pas établi de règles systématiques de construction de ces fichiers car nous pensons qu'elles dépendent du mode de stockage de données possibles, matière que nous n'avons pas approfondie.
Quant aux fichiers logiques représentant la mémoire, ils se transformeront également en fichiers réels selon les règles suivantes :

- les entités seront implémentées par des fichiers réels dont les articles contiennent des identifiants pouvant former la clé d'accès, ainsi que les attributs de l'entité;
- les associations contenant des attributs seront implémentées par des fichiers réels dont les articles contiennent les identifiants des entités que l'association relie, et les attributs de l'association;

- les associations ne contenant pas d'attributs ne seront pas implémentées et pourront être retrouvées grâce aux références dans les entités.

Le deuxième type de fichier physique est le fichier virtuel qui opérera une distribution dans le temps du fichier réel.

En appliquant cette transformation à la figure 3.14., on obtient la figure 3.15.

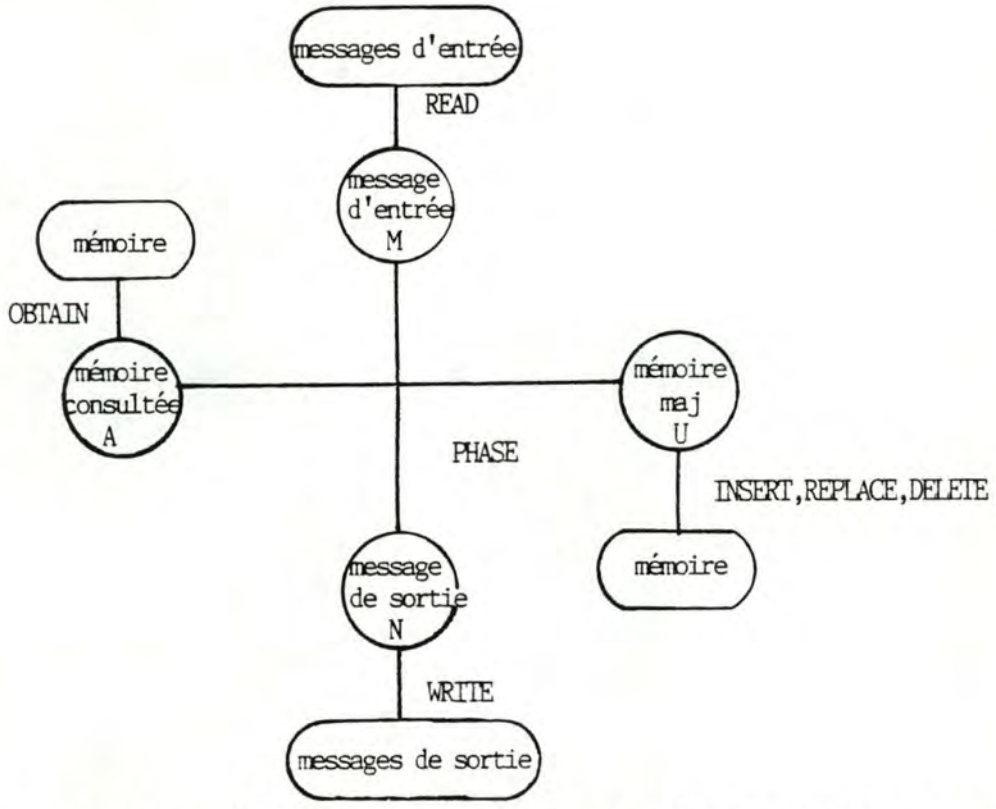


figure 3.15. Modèle Memo-Proges de l'architecture des données au niveau d'une phase.

L'accès aux informations n'est pas réglé au niveau de la phase, mais sera réglé au niveau de ses fonctions par les modules d'entrée-sortie au § 3.3.2.2.1.B.2°.

3° Exemple.

A titre d'exemple, en appliquant ces transformations à la phase ENREGISTREMENT-BON-DE-COMMANDE-CLIENT, on obtient le schéma Memo-Proges de représentation des fichiers de la figure 3.16.

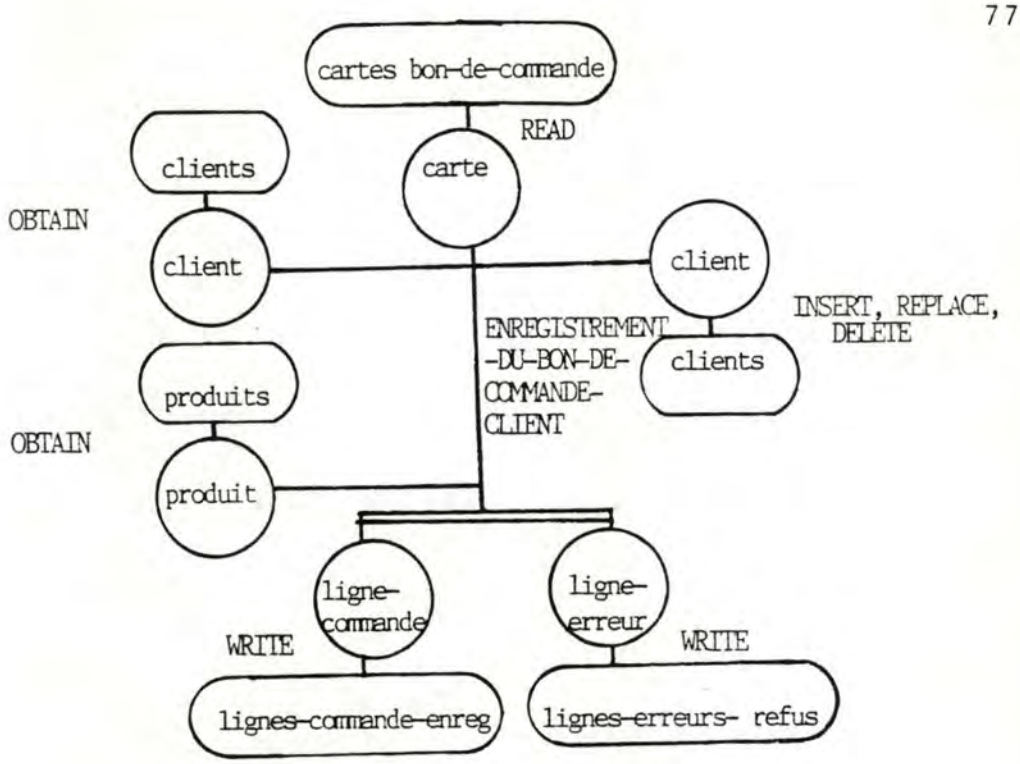


figure 3.16. Modèle Memo-Proges de la phase ENREGISTREMENT-DU-BON-DE-COMMANDE-CLIENT .

4° Spécification des fichiers virtuels et réels.

La spécification des fichiers réels et virtuels consistera en la description de chacun des types d'articles de ces fichiers. Cette description contiendra par type d'article la description de ses éléments et leur format. Tous ces renseignements peuvent être trouvés dans la spécification des informations faite en D.S.L. lors de l'analyse conceptuelle.

A titre d'exemple, le fichier CLIENT pourra être spécifié à partir de la spécification D.S.L. de l'entité client et de ses éléments :

```
DEFINE ENTITY client;
   DESCRIPTION;
      est considéré comme client, toute
      personne physique ou morale qui a
      au moins passé à la firme une com-
      mande qui a permis de l'identifier;
   IDENTIFIED BY numéro-de-client;
   RELATED    BY client/commande;
   CONSISTS   OF numéro-de-client;
   CONSISTS   OF nom-de-client;
   CONSISTS   OF adresse-du-client;
   CARDINALITY IS 40 000;
```

```
DEFINE ELEMENT numéro-de-client;
   DESCRIPTION;
      Code attribué par la firme à un nouveau
      client.  Ce code doit permettre d'identifier
      le client sans ambiguïté;
   FORMAT IS X(6);
   IDENTIFIES client;

DEFINE ELEMENT nom-de-client;
   DESCRIPTION;
      Nom sous lequel le client est connu de la firme
      pour toutes les transactions le concernant;
   FORMAT IS X(30);

DEFINE ELEMENT adresse-du-client;
   DESCRIPTION;
      Adresse à laquelle la firme envoie les différents
      documents et les colis pour un client;
   FORMAT IS X(60);
```

On obtient ainsi la description de l'article client suivante :

```
*** CLI : SIGNALETIQUE CLIENT ***
 21 NUMERO-DE-CLIENT  PIC X(6).
 21 NOM-DE-CLIENT     PIC X(30).
 21 ADRESSE-DU-CLIENT PIC X(60).
```

3.3.2.2. Architecture des traitements.

L'architecture des traitements sera construite au niveau de chaque phase comme une réinterprétation organique en Memo-Proges du modèle statique des fonctions de la phase et du modèle dynamique exprimant leur enchaînement. Une fois l'architecture réalisée pour chaque phase, il resterait à les enchaîner. Ce problème n'étant pas du ressort de Memo-Proges (1), nous ne le traiterons pas dans le cadre de ce mémoire.

3.3.2.2.1. Transformation du modèle statique en modèle Memo-Proges.

A. Modèle statique.

Le modèle conceptuel de la statique d'une fonction est représenté par la figure 3.17.

---

(1) La résolution de ce problème est liée au langage de commande de l'ordinateur utilisé.

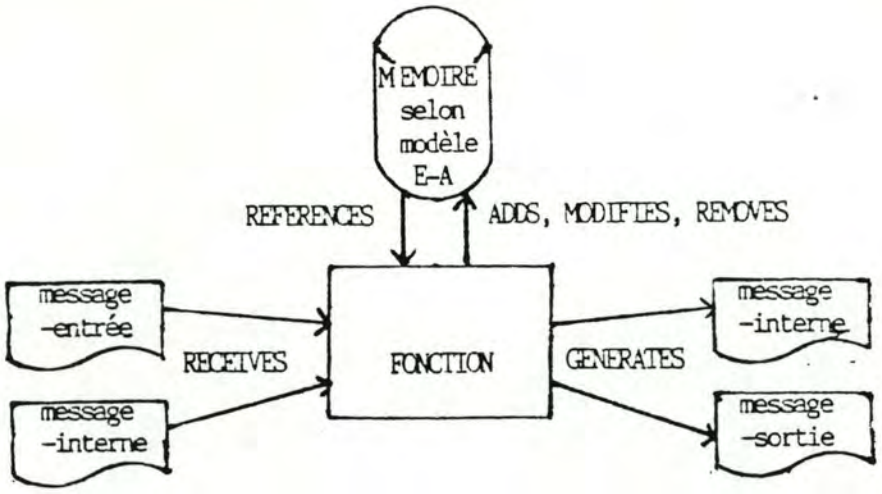


figure 3.17. Modèle statique d'une fonction.

B. Règles de passage.

Nous suggérons les transformations suivantes :

1° Transformation de la fonction.

A une fonction identifiée lors de l'analyse conceptuelle, correspondra un module fonctionnel.

Ainsi les fonctions composant la phase "ENREGISTREMENT-DU-BON-DE-COMMANDE-CLIENT" se transforment en modules fonctionnels de la manière suivante :

AFFICHER <résultat-contrôle-identité>	devient	CTRLID
AFFICHER <résultat-contrôle-ligne>		CTRLIG
AFFICHER <résultat-contrôle-total>		CTRTOT
AJOUTER <client>		ADDCLI
MODIFIER <client>		MODCLI
AJOUTER <ligne-commande>		ADLCMD
EDITER <justificatif-refus>		CTRERR

2° Transformation de l'accès aux informations.

L'ensemble des messages en entrée de la fonction (message d'entrée et message interne) et l'ensemble des messages en sortie de la fonction (message interne et message de sortie) deviennent des fichiers virtuels respectivement de statut M et N. Le contenu d'un article de ces fichiers virtuels est équivalent au contenu du message correspondant.

Quant aux éléments de la mémoire consultée ou mise à jour, ils se transforment en fichiers virtuels respectivement de statut A et U. Pour obtenir la description d'un article de ce fichier, il suffira de regrouper l'ensemble des éléments consultés ou mis à jour par entité ou association.

L'accès aux informations est réalisé par le couplage des modules d'entrée-sortie aux modules fonctionnels (§ 2.2.2.2.1°.).

C. Modèle Memo-Proges.

A partir du modèle statique de la figure 3.17., et en appliquant les règles de transformation que nous venons de proposer, nous obtenons le schéma Memo-Proges de la figure 3.18.

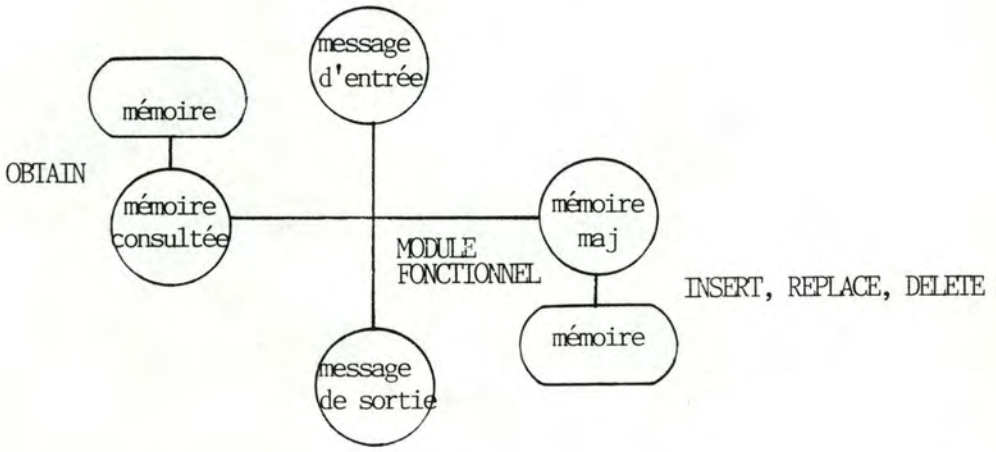


figure 3.18. Modèle Memo-Proges correspondant au modèle statique d'une fonction.

D. Exemple.

A titre d'exemple, la fonction AFFICHER résultat-contrôle-identité représentée lors de l'analyse conceptuelle selon la figure 3.19., peut se transformer en le schéma Memo-Proges de la figure 3.20.

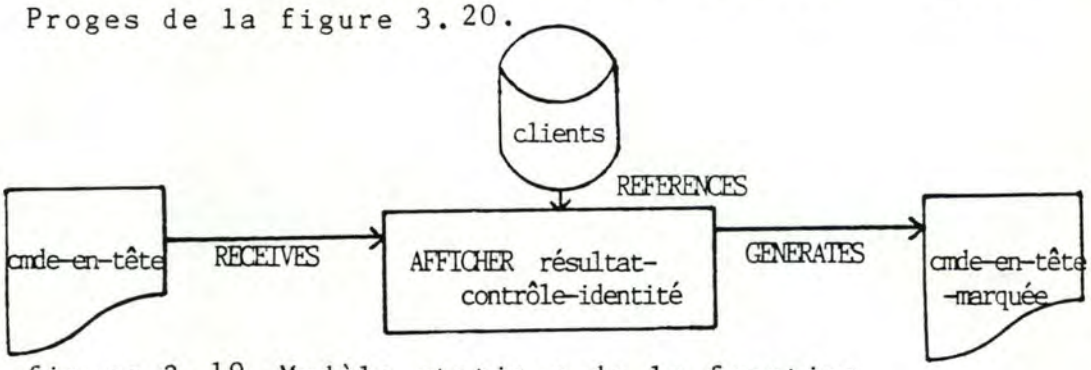


figure 3.19. Modèle statique de la fonction AFFICHER résultat-contrôle-identité .

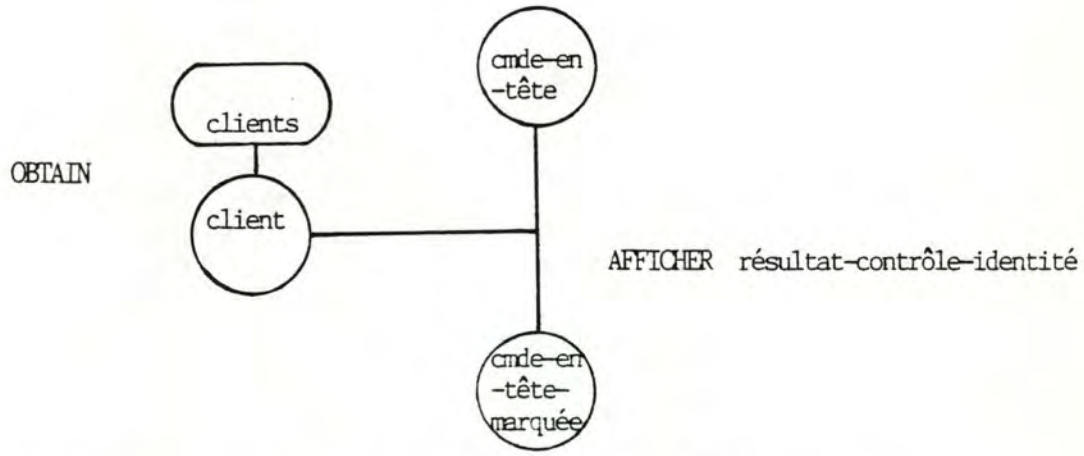


figure 3.20. Modèle Memo-Proges de la fonction AFFICHER résultat-contrôle-identité .

3.3.2.2.2. Transformation du modèle dynamique en modèle Memo-Proges.

A. Principes de transformation.

Les problèmes avec flux d'information ont été définis au § 1.3.1.1. comme des suites de transformations appliquées à chaque message d'entrée pour obtenir un message de sortie.

Cette suite de transformations a été modélisée à l'aide des concepts de processus, d'événements et des mécanismes de synchronisation, de condition et d'itération du modèle dynamique de [Bodart-Pigneur]. Les processus représentent les exécutions des fonctions. Les événements internes et les mécanismes du modèle permettent d'enchaîner les fonctions.
L'enchaînement des fonctions peut dès lors être modélisé dans le modèle de Bodart comme à la figure 3.21. où le point de synchronisation S est considéré au sens large et représente un des 6 enchaînements dynamiques élémentaires.

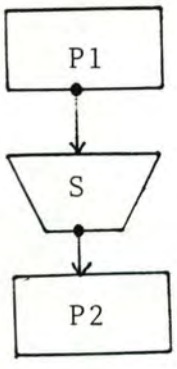


figure 3.21. Modèle dynamique d'enchaînement des fonctions.

Un cas particulier est le déclenchement de la première fonction de la phase par un événement externe "réception d'un message" représenté à la figure 3.22.

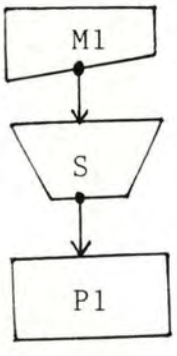


figure 3.22. Modèle dynamique de déclenchement d'une fonction.

Nous proposons de transformer le modèle dynamique d'enchaînement des fonctions en modèle Memo-Proges de couplage de modules comme le montre la figure 3.23..

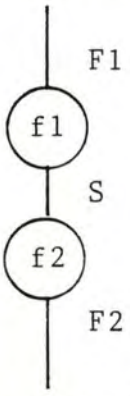


figure 3.23. Modèle Memo-Proges d'enchaînement des fonctions.

Les règles de transformation sont les suivantes :

- aux processus P1 et P2 correspondent les modules fonctionnels F1 et F2;
- au point de synchronisation correspond 0 ou 1 module coordinateur (1);
- à l'événement interne "terminaison du processus P1" correspond le message f1;
- à l'événement interne "réalisation du point de synchronisation S" correspond le message f2;

Quant au modèle de déclenchement d'une fonction par un événement externe, nous proposons de le transformer en modèle Memo-Proges tel que représenté à la figure 3.24..

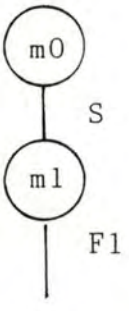


figure 3.24. Modèle Memo-Proges de déclenchement d'une fonction.

Les règles de transformation sont les suivantes :

- au processus P1 correspond le module fonctionnel F1;

(1) La présence ou non d'un module coordinateur dépend de l'enchaînement des fonctions (cf point suivant B.).

- au point de synchronisation correspond 0 ou 1 module coordinateur;
- à l'événement externe "réception du message M1" correspond le message m0;
- à l'événement interne "réalisation du point de synchronisation S" correspond le message m1;

Pour opérer cette transformation, il faut d'une part pouvoir transformer les 6 enchaînements de base en modèle de couplage de modules Memo-Proges (ce sera l'objet du point B) et d'autre part pouvoir définir le contenu des messages f1 et f2.
Le message f1 représente la sortie du module fonctionnel F1 et donc l'entrée du module de synchronisation S.
[Bodart-Pigneur] définit :

- un modèle statique pour les fonctions qui permet de retrouver les messages de sortie de F1 et les messages d'entrée de F2, et
- un modèle dynamique exprimant que S relie F1 et F2.

On peut donc induire la règle suivante : "le module de synchronisation S "transforme" le message f1 en message f2".
De deux choses l'une

soit f1 = f2 alors il n'y a pas de règles de transformation de messages à spécifier,
soit f1 ≠ f2 ce qui signifie que le point de synchronisation n'est pas fonctionnellement neutre et il faut donc lui donner des spécifications fonctionnelles.

B. Transformation des enchaînements de base en modèle Memo-Proges.

1° Enchaînement séquentiel.

Un enchaînement est dit séquentiel lorsque la terminaison d'un processus-fonction 1 représentée par un point, provoque le déclenchement d'un processus-fonction 2. Un enchaînement séquentiel est présenté à la figure 3.25..

Nous suggérons la transformation suivante :
A l'événement "terminaison du processus-fonction 1" correspond le message interne sortant de la fonction 1. L'enchaînement séquentiel se transforme ainsi en couplage de modules correspondant à la fonction 1 et à la fonction 2. Ce couplage est réalisé par l'interface fichier virtuel représentant les messages internes sortant de la fonction 1.

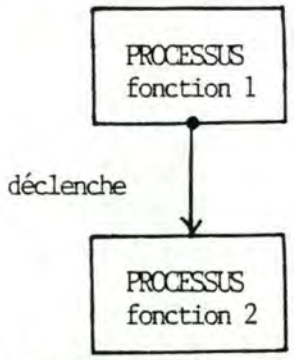


figure 3.25. Schéma de l'enchaînement séquentiel [Bodart-Pigneur].

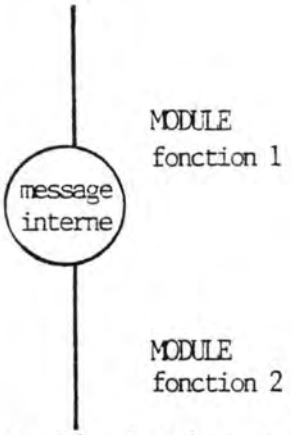


figure 3.26. Schéma de l'enchaînement séquentiel Memo-Proges.

Nous présentons, à titre d'exemple la transformation en schéma Memo-Proges de l'enchaînement dynamique des fonctions de la phase CORRECTION dont la spécification a été donnée au § 3.2.2.2.4..
Les figures 3.27. et 3.28. représentent l'enchaînement selon les 2 représentations.

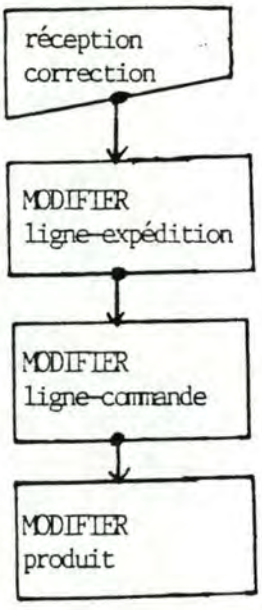


figure 3.27. Phase CORRECTION schéma dynamique [Bodart-Pigneur].

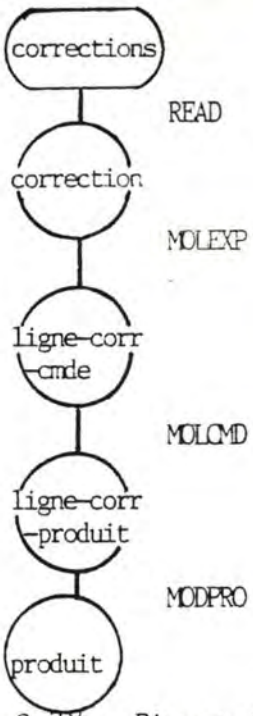


figure 3.28. Phase CORRECTION schéma dynamique Memo-Proges.

2° Enchaînement éclaté.

Un enchaînement est dit éclaté lorsque la terminaison d'un processus-fonction 1 provoque le déclenchement simultané des processus-fonction 2, processus-fonction 3, ... , processus-fonction n. Un enchaînement éclaté est présenté à la figure 3.29.
Nous suggérons la transformation suivante :
A l'événement "terminaison du processus-fonction 1" correspond le message interne sortant de la fonction 1.
Aux processus-fonction i correspondent les modules-fonction i.
La relation de déclenchement simultané de processus est transformée en appels séquentiels de modules.
Ces appels seront réalisés dans un module coordinateur d'éclatement (LINK). Ce module coordinateur sera couplé aux modules fonctionnels par l'interface fichier virtuel "messages sortant de la fonction 1" équivalent aux "messages entrant dans les fonction 2, fonction 3, ... , fonction n".

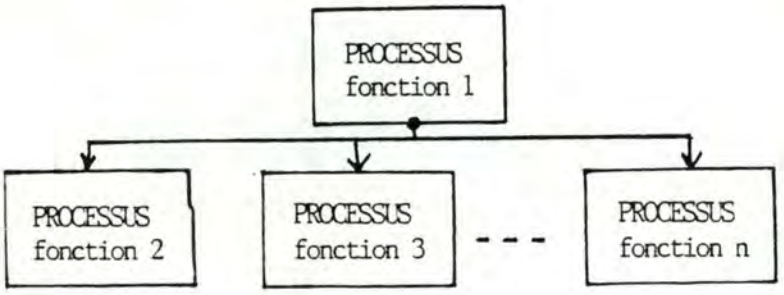


figure 3.29. Schéma de l'enchaînement éclaté [Bodart-Pigneur].

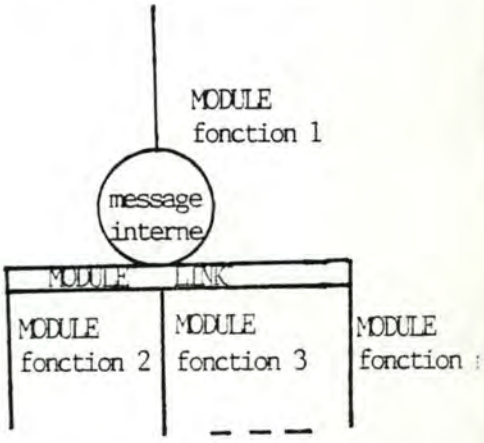


figure 3.30. Schéma de l'enchaînement éclaté Memo-Proges.

Nous présentons, à titre d'exemple la transformation en schéma Memo-Proges de l'enchaînement dynamique de la phase MAJ-MOINS-DU-STOCK dont la spécification a été donnée au § 3.2.2.2.2..
Les figures 3.31. et 3.32. représentent l'enchaînement selon les 2 représentations.

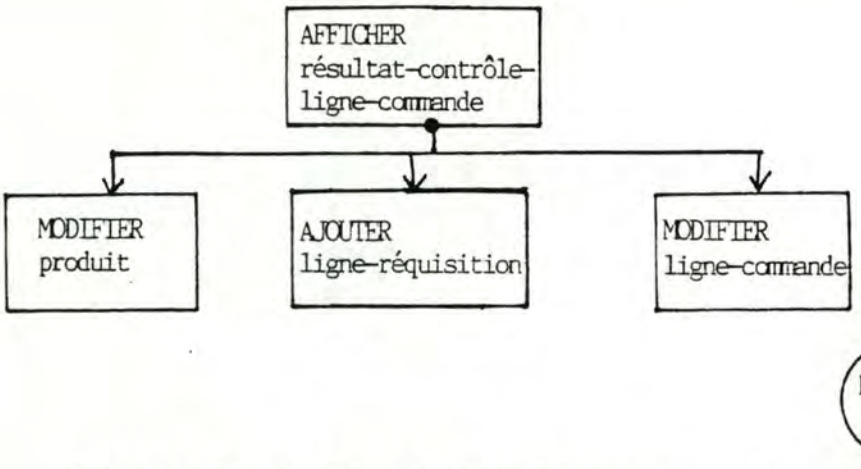


figure 3.31. Phase MAJ-MOINS-DU-STOCK
schéma dynamique
[Bodart-Pigneur].

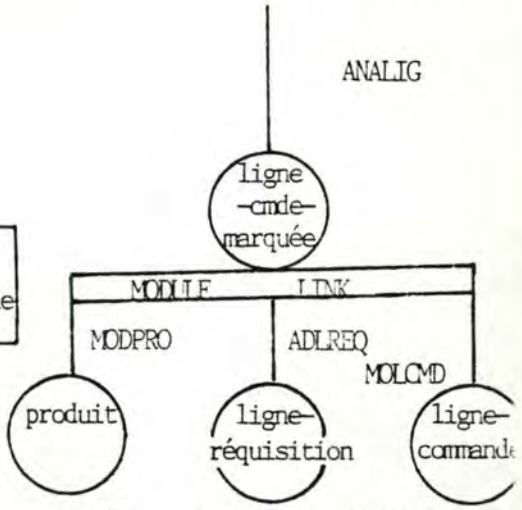


figure 3.32. Phase MAJ-MOINS-DU-STOC
schéma dynamique
Memo-Proges.

3° Enchaînement multiple.

Un enchaînement est dit multiple lorsque la terminaison d'un processus-fonction 1 provoque le déclenchement simultané de n (> 1) processus-fonction 2, c'est-à-dire de n exécutions de la fonction 2.
La transformation de cette structure sous forme d'appels de modules nécessite l'ajoût d'un module coordinateur de déclenchement multiple (MDM). Le module-fonction 1 appelle le module MDM en lui transmettant dans l'interface le message sortant de la fonction 1 et une information lui permettant de déterminer le nombre d'appels à effectuer. Le module MDM effectue alors les appels successifs au module-fonction 2 en lui transmettant dans l'interface le message entrant de la fonction 2.

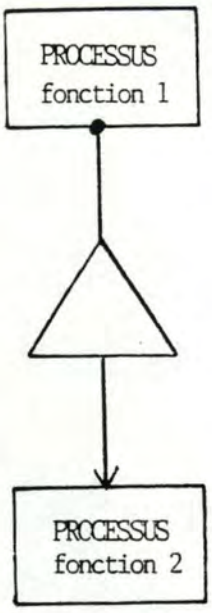


figure 3.33. Schéma de l'enchaînement multiple [Bodart-Pigneur].

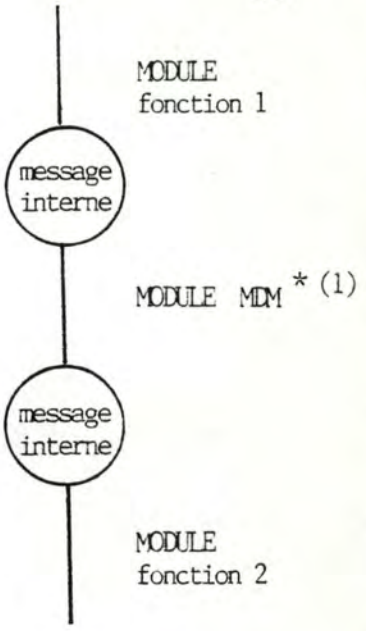


figure 3.34. Schéma de l'enchaînement multiple Memo-Proges.

Soit à titre d'exemple, le début de la phase ENREGISTREMENT-DU-BON-DE-COMMANDE-CLIENT, où l'on reçoit une commande complète qui est éclatée en lignes devant chacune subir un contrôle. La spécification de cette phase est donnée au § 3.2.2.2.1..
Les figures 3.35. et 3.36. représentent l'enchaînement selon les 2 représentations.

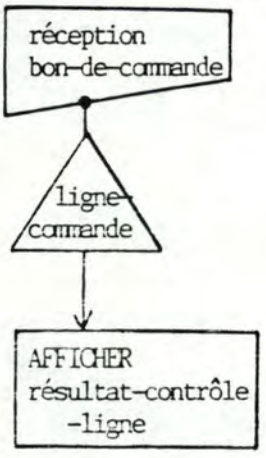


figure 3.35. Phase ENREGISTREMENT-DU-BON-DE-COMMANDE-CLIENT schéma dynamique [Bodart-Pigneur].

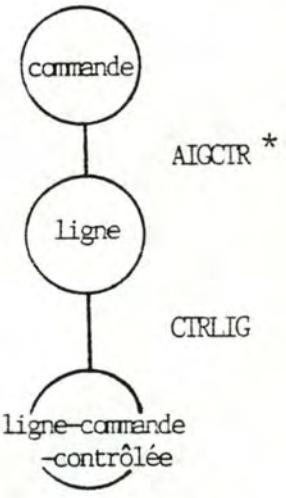


figure 3.36.Phase ENREGISTREMENT-DU-BON-DE-COMMANDE-CLIENT Schéma dynamique Memo-Proges.

(1) Memo-Proges ne possède pas de convention pour représenter le déclenchement multiple. Nous proposons la représentation * à l'instar de Jackson.

4° Enchaînement conditionnel.

Un enchaînement est dit conditionnel lorsque la terminaison d'un processus-fonction 1 déclenche le processus-fonction 2 si la condition C1 est vraie et déclenche le processus-fonction 3 si la condition C1 est fausse.
La transformation de cette structure nécessite l'ajoût d'un module coordinateur de test de la condition d'éclatement (MTC). Le module-fonction 1 appelle le module MTC en lui transmettant dans l'interface le message sortant de la fonction 1.
Le module MTC teste le prédicat associé à la condition. S'il est vrai, MTC appelle le module-fonction 2 en lui transmettant dans l'interface le message entrant dans la fonction 2. S'il est faux, MTC appelle le module-fonction 3 en lui transmettant dans l'interface le message entrant dans la fonction 3.

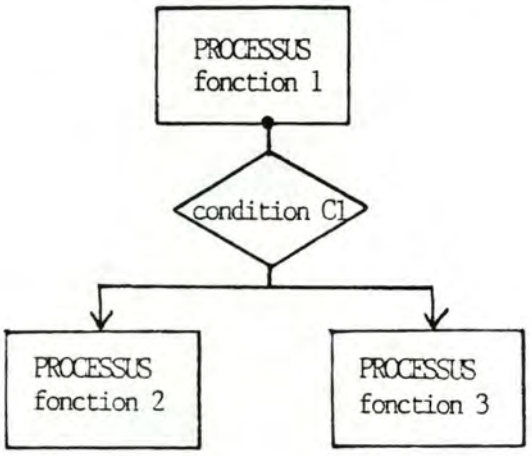


figure 3.37. Schéma de l'enchaînement conditionnel [Bodart-Pigneur].

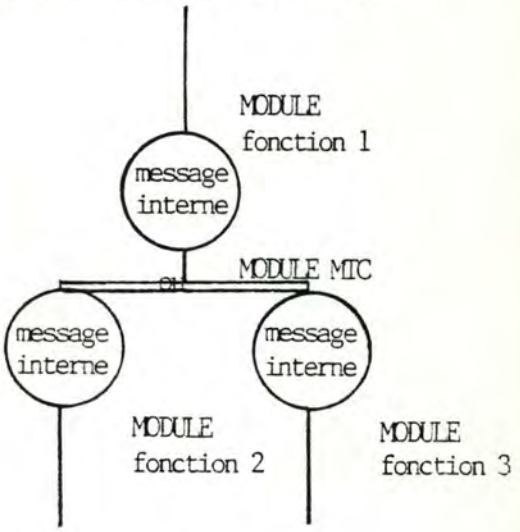


figure 3.38. Schéma de l'enchaînement conditionnel Memo-Proges

Soit à titre d'exemple, la phase ARCHIVAGE dont la spécification a été donnée au § 3.2.2.2.6..
Les figures 3.39. et 3.40. représentent l'enchaînement selon les 2 représentations.

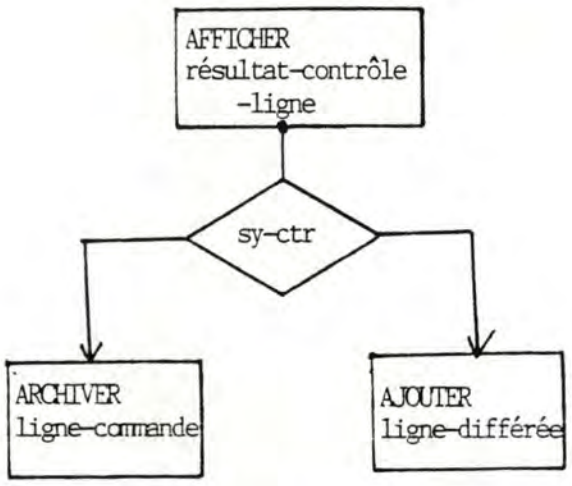


figure 3.39. Phase ARCHIVAGE
schéma dynamique [Bodart-Pigneur].

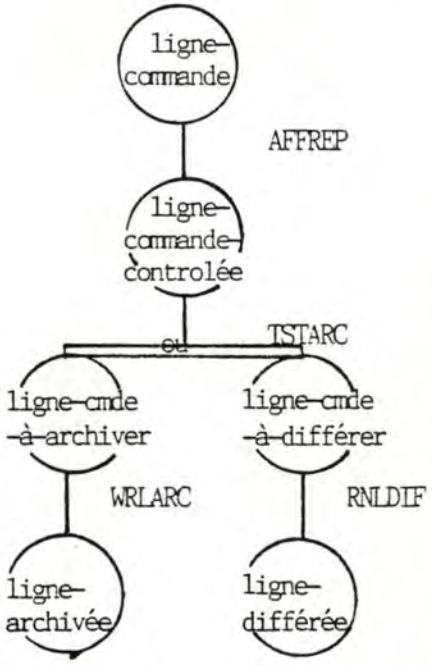


figure 3.40: Phase ARCHIVAGE
schéma dynamique Memo-Proges.

5° Enchaînement convergent.

Un enchaînement est dit convergent si le déclenchement du processus-fonction i est provoqué par la terminaison d'un des processus-fonction 1 ou processus-fonction 2 ... ou processus-fonction n.
On notera que sous l'angle d'un des événements déclencheurs, cette structure ne se différencie pas de la structure séquentielle. En effet, chaque événement agit indépendamment des autres événements déclencheurs éventuels. On transformera donc cette structure en n structures séquentielles.

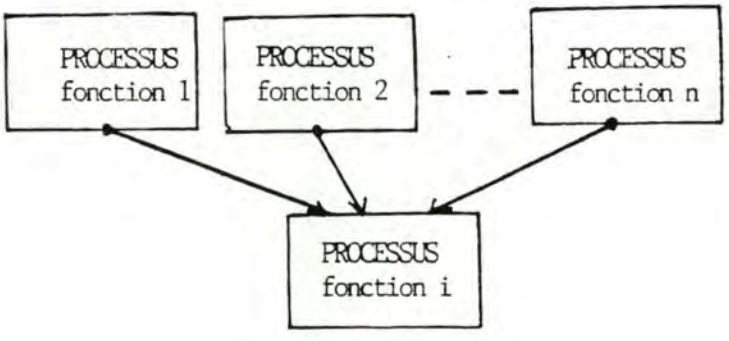


figure 3.41. Schéma de
l'enchaînement convergent
[Bodart-Pigneur].

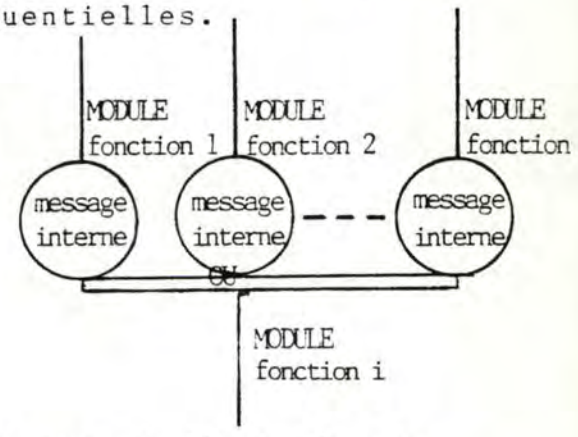


figure 3.42. Schéma de
l'enchaînement convergent
Memo-Proges.

Nous n'avons pas rencontré d'exemple d'enchaînement convergent dans le cas PETITPAS. Cependant, pour illustrer cette transformation, nous prendrons l'exemple suivant :

"L'impression d'un extrait de compte bancaire peut être déclenchée par la terminaison de l'enregistrement d'un paiement ou par la terminaison de l'enregistrement d'un versement."

Les figures 3.43. et 3.44. représentent l'enchaînement selon les 2 représentations.

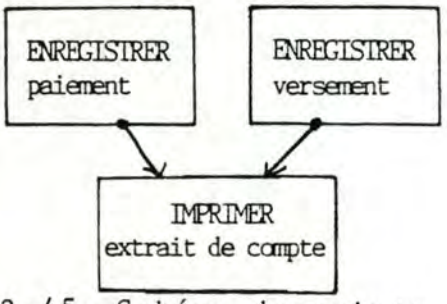


figure 3.45. Schéma dynamique [Bodart-Pigneur].

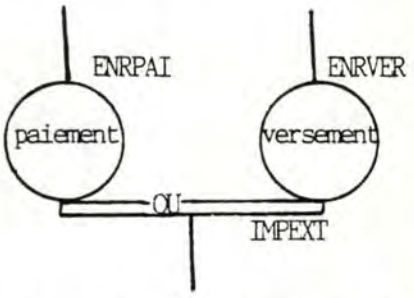


figure 3.46. Schéma dynamique Memo-Proges.

6° Enchaînement synchronisé.

L'enchaînement synchronisé introduit le mécanisme de coordination d'événements le "point de synchronisation".

On peut distinguer 2 types de points de synchronisation selon qu'ils coordonnent différents types d'événements (structure parallèle convergente) ou n occurrences du même type d'événement (structure d'accumulation). Nous allons présenter successivement la transformation de ces 2 structures.

Dans la structure parallèle convergente, la terminaison des processus-fonction 1 et processus-fonction 2 agissent sur la valeur du prédicat CS1. Lorsque celui-ci est vrai, le processus-fonction 3 est déclenché. CS1 représente en fait une condition définissant une attente : l'attente imposée aux événements "terminaison du processus-fonction 1" ou "terminaison du processus-fonction 2" avant qu'ils ne produisent leurs effets dynamiques.

La transformation de cette structure nécessite l'ajoût d'un module coordinateur de synchronisation (MCS). Les module-fonction 1 et module-fonction 2 appellent le module MCS en lui transmettant dans l'interface le message sortant respectivement de la fonction 1 et de la fonction 2. MCS teste la valeur du prédicat associé à la condition de synchronisation. Lorsque le prédicat est vrai, le module MCS, avant de retourner au module appelant, appelle le module-fonction 3 en lui transmettant dans l'interface le message correspondant à la "réalisation du point de synchronisation" équivalent au message entrant dans la fonction 3 et réinitialise son attente. Lorsque le prédicat est faux, le module MCS mémorise éventuellement le message avant de retourner au module appelant.

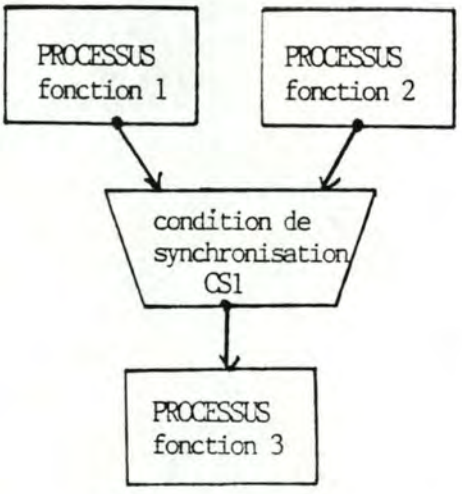


figure 3.47. Schéma de
l'enchaînement parallèle convergent
[Bodart-Pigneur].

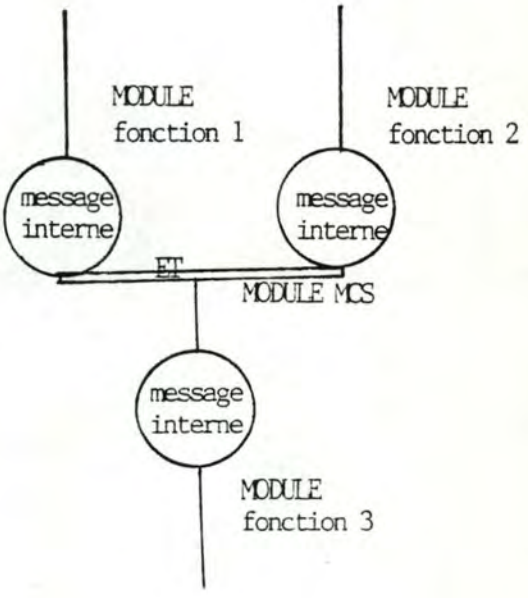


figure 3.48. Schéma de
l'enchaînement parallèle convergent
Memo-Proges.

Dans la structure d'accumulation, la condition de synchronisation CS1 porte sur la survenance de n (> 1) occurrences d'un même type d'événement. Les terminaisons successives du processus-fonction 1 participent au point de synchronisation CS1.
La transformation de cette structure en appels de modules nécessite également l'ajoût d'un module coordinateur d'accumulation (MCA). Les exécutions successives du module-fonction 1 appellent chacune MCA en lui transmettant dans l'interface le message de sortie de la fonction 1. MCA teste le nombre d'appels qu'il attend. Lorsque ce nombre atteint la valeur de n, MCA appelle le module-fonction 2 en lui transmettant dans l'interface le message correspondant à l'événement "réalisation du point de synchronisation".

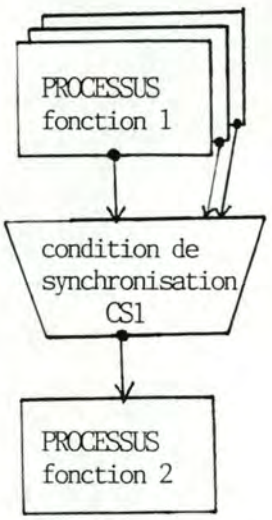


figure 3.49. Schéma de l'enchaînement d'accumulation [Bodart-Pigneur].

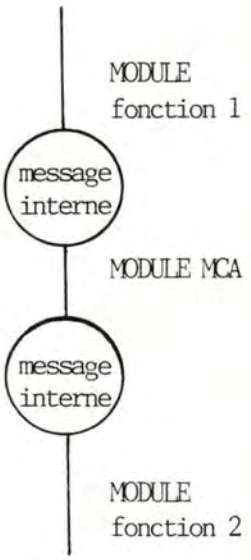
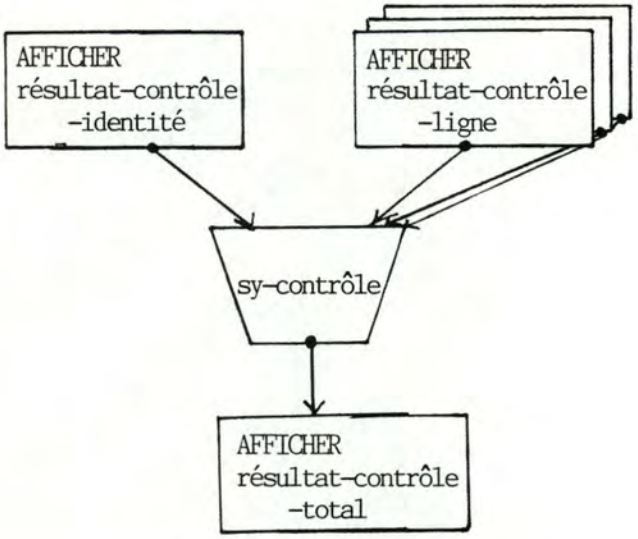


figure 3.50. Schéma de l'enchaînement d'accumulation Memo-Proges.

A titre d'exemple, la phase ENREGISTREMENT-DU-BON-DE-COMMANDE-CLIENT dont la spécification a été donnée au § 3.2.2.2.1. présente un point de synchronisation réalisant à la fois la convergence entre les contrôles de l'identité d'une commande et de ses lignes, et d'autre part l'accumulation des lignes de cette commande après leur contrôle.



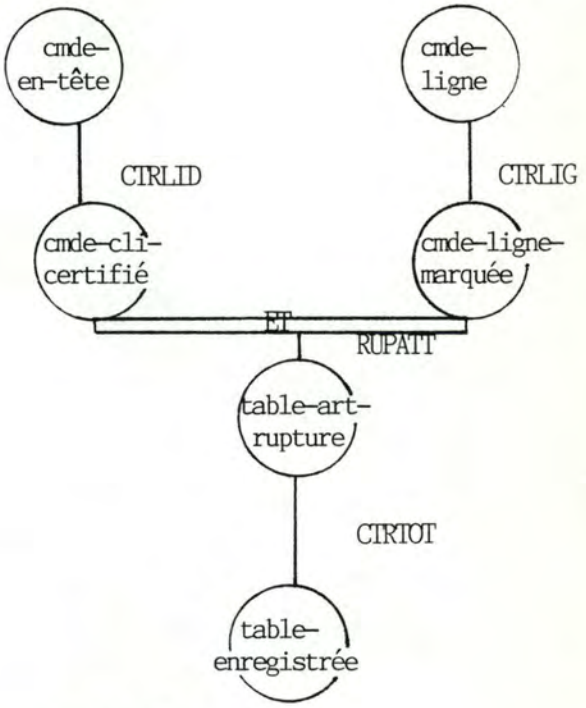


figure 3.51. Phase ENREGISTREMENT-BON-DE-COMMANDE-CLIENT
schéma dynamique [Bodart-Pigneur].

figure 3.52. Phase ENREGISTREMENT-BON-COMMANDE-CLIENT
schéma dynamique Memo-Proges.

C. Exemple.

Pour illustrer la transformation du modèle dynamique des fonctions, nous prendrons l'exemple de la phase ENREGISTREMENT-BON-DE-COMMANDE-CLIENT du cas PETITPAS.
La figure 3. 53. représente le schéma du modèle dynamique défini lors de l'analyse conceptuelle au niveau des fonctions.
La figure 3. 54. montre la transformation de ce schéma en Memo-Proges en appliquant les règles que nous venons d'énoncer.

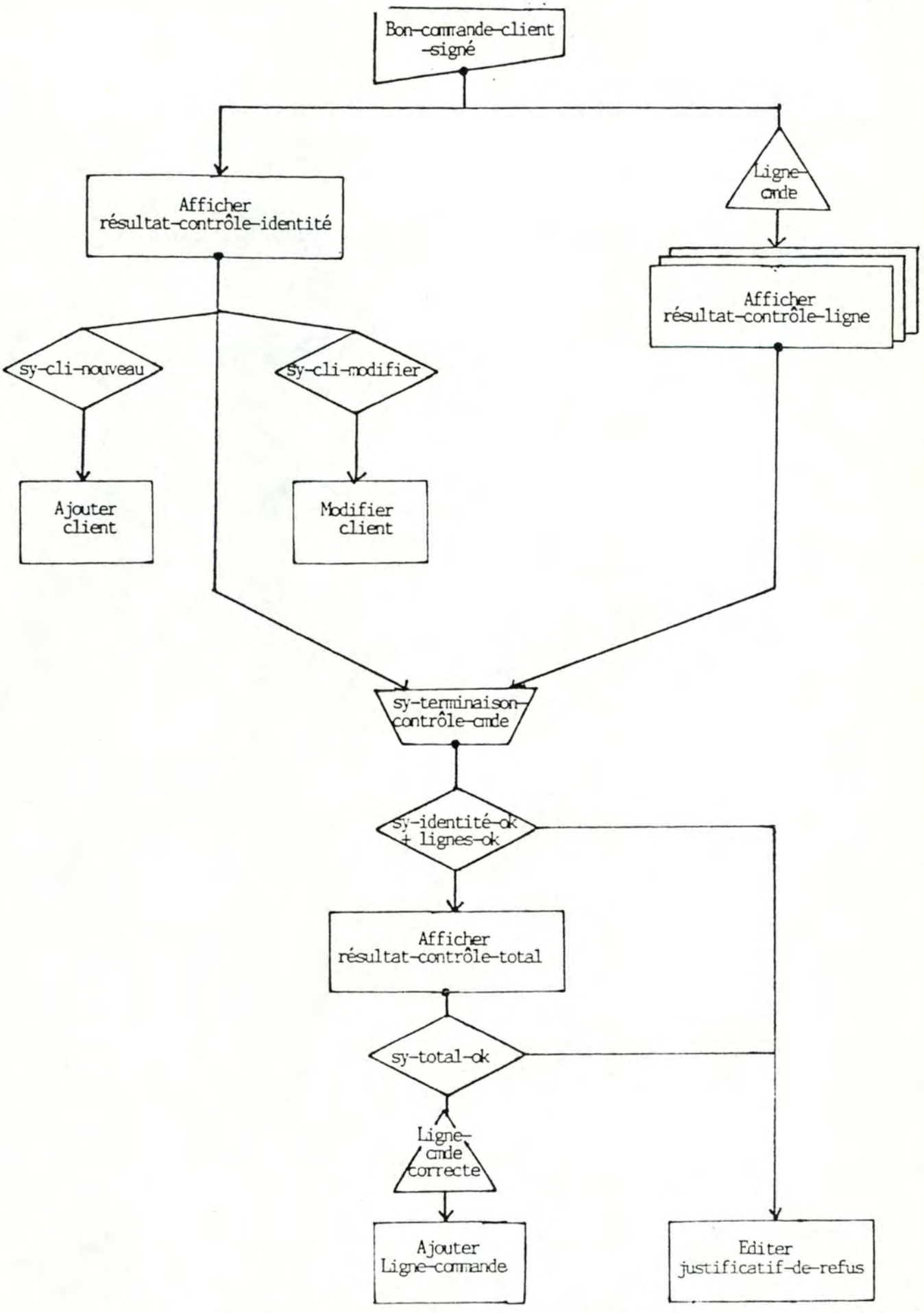


figure 3.53. Dynamique de la phase "ENREGISTREMENT-BON-DE-COMMANDE-CLIENT".

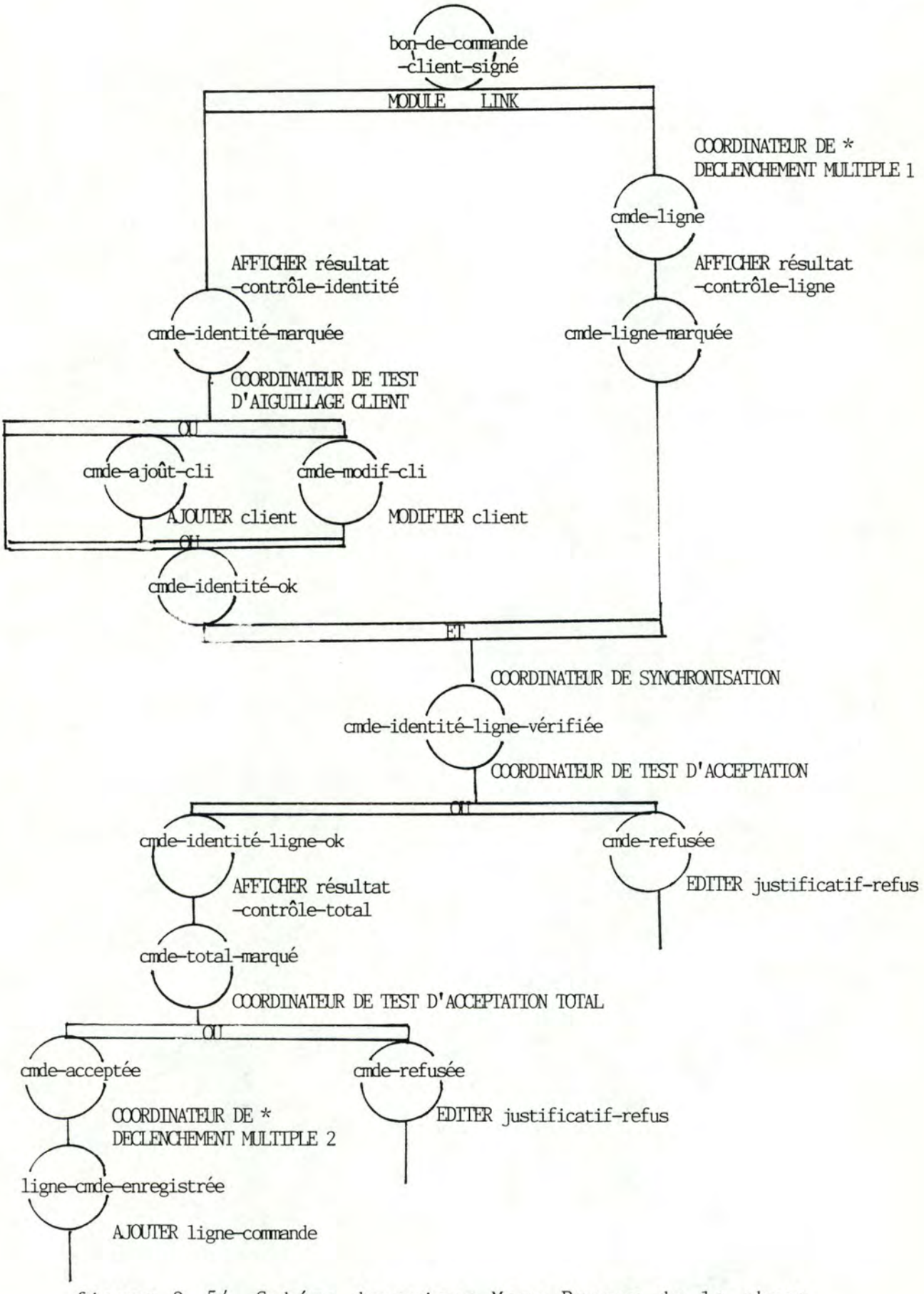


figure 3.54. Schéma dynamique Memo-Proges de la phase "ENREGISTREMENT-BON-DE-COMMANDE-CLIENT".

3.3.2.2.3. Spécification des modules.

Une fois tous les modules identifiés, ils doivent être spécifiés. Les techniques de spécification de modules sont nombreuses, cependant nous en retiendrons trois comme principales :

- les spécifications par règles, expression libre des règles de transformation;
- les spécifications par assertions, pré- et post-conditions;
- les spécifications algébriques, types abstraits de [Liskov].

Nous avons choisi de procéder par règles. Nous donnerons les spécifications des modules suivant les 3 classes définies au § 2.2.2.2.,c'est-à-dire les modules fonctionnels, les modules coordinateurs et les modules d'entrée-sortie.

A. Spécification des modules fonctionnels.

Pour la spécification des modules fonctionnels, nous reprendrons la spécification de l'analyse conceptuelle de la manière suivante :

DEFINE PROCESS nom-de-fonction;	MODULE nom-de-fonction
RECEIVES nom-info1;	ENTREE nom-info1
GENERATES nom-info2;	SORTIE nom-info2
REFERENCES nom-info3;	CONSULTE nom-info3
ADDS / MODIFIES / REMOVES nom-info4;	AJOUTE / MODIFIE / SUPPRIME nom-info4
DESCRIPTION; ... ;	FONCTION ...

B. Spécification des modules coordinateurs.

Les modules coordinateurs proviennent de la transformation des 6 structures dynamiques de [Bodart-Pigneur] à l'aide desquelles on peut définir toute dynamique. Leurs spécification peuvent donc être standardisées de la manière suivante :

Module coordinateur d'éclatement :

La spécification standardisée du module LINK (figure 3.30.) est la suivante :

argument d'entrée : message sortant du module-fonction1

fonction : appelle les modules -fonction2, -fonction3, ...

argument de sortie : message sortant du module-fonction1

A titre d'exemple, nous spécifierons le module d'éclatement LINK de la phase ENREGISTREMENT-BON-DE-COMMANDE-CLIENT :

argument d'entrée : bon-commande-client-signé

fonction : appelle le module AFFICHER résultat-contrôle-ientité et le module COORDINATEUR DE DECLENCHEMENT MULTIPLE.

argument de sortie : bon-commande-client-signé.

Module coordinateur de déclenchement multiple :

La spécification standardisée du module MDM (figure 3.34.) est la suivante :

argument d'entrée : message sortant du module-fonction1 + information lui permettant de déterminer le nombre d'appels n à effectuer

fonction : détermine le nombre d'appels n à effectuer et appelle n fois le module-fonction2

argument de sortie : message entrant dans module-fonction2

Ainsi le module COORDINATEUR DE DECLENCHEMENT MULTIPLE 1 de la phase ENREGISTREMENT-BON-DE-COMMANDE-CLIENT est spécifié de la manière suivante :

argument d'entrée : bon-commande-client-signé + le nombre de lignes de la commande (= le nombre d'appels à exécuter)

fonction : appelle "nbre-de-lignes-cmde" fois le module AFFICHER résultat-contrôle-ligne

argument de sortie : cmde-ligne

Module coordinateur de test de la condition d'éclatement :

La spécification standardisée du module MTC (figure 3.38.) est la suivante :

argument d'entrée : message sortant du module-fonction1

fonction : calcule la valeur du prédicat C1
- s'il est vrai, appelle le module-fonction2
- s'il est faux, appelle le module-fonction3

argument de sortie : -soit message entrant dans module-fonction2
-soit message entrant dans module-fonction3

A titre d'exemple, nous spécifierons le module de test COORDINATEUR DE TEST D'AIGUILLAGE CLIENT de la phase ENREGISTREMENT-BON-DE-COMMANDE-CLIENT :

argument d'entrée : cmde-identité-marquée

fonction : si marque =1 alors appeler module MODIFIER client
si marque =2 alors appeler module AJOUTER client
si (marque ≠1) et (marque ≠2)
alors appeler module COORD SYNCHRO

argument de sortie : cmde-modif-cli
cmde-ajoût-cli
cmde-identité-ok

Module coordinateur de synchronisation :

La spécification standardisée du module MCS (figure 3.48.) est la suivante :

argument d'entrée : soit message sortant du module-fonction1
soit message sortant du module-fonction2

fonction : teste la valeur du prédicat de synchronisation CS1
- s'il est vrai, appelle le module-fonctiion3 et réinitialise l'attente
- s'il est faux, mémorise éventuellement le message d'entrée et attend

argument de sortie : message entrant dans module-fonction3

Ainsi, le module COORDINATEUR SYNCHRONISATION de la phase ENREGISTREMENT-BON-DE-COMMANDE-CLIENT est spécifié de la manière suivante :

argument d'entrée : cmde-ident-ok
cmde-ligne-marquée

fonction : teste la valeur du prédicat "terminaison-vérification-contrôle-ligne"
tant que le prédicat est faux (pas de rupture sur le numéro-commande), mémorise le message et attend
dès que le prédicat devient vrai (rupture sur le numéro-commande), on appelle le module COORDINATEUR DE TEST D'ACCEPTATION

argument de sortie : cmde-identité-ligne-vérif

Module coordinateur d'accumulation :

La spécification standardisée du module MCA (figure 3.50.) est la suivante :

argument d'entrée : message sortant du module-fonction1

fonction : déterminer si le nombre d'appels = n
- si oui, appelle le module-fonction2
- si non, incrémente de 1 le nombre d'appels

argument de sortie : message entrant dans module-fonction2

Il n'y a pas d'exemple de module coordinateur d'accumulation dans la phase ENREGISTREMENT-BON-DE-COMMANDE-CLIENT.

C. Spécification des modules d'entrée-sortie.

Etant donné que les modules d'entrée-sortie sont standardisés, leurs spécifications se réduisent à l'expression de :

- l'opération d'accès à effectuer (exprimant le mode d'accès logique). Les opérations d'accès possibles ont été exposées au § 2.2.1.1.3° selon le tableau de la figure 3.55. :

M	READ	N	WRITE
A	OBTAIN	U	INSERT REPLACE DELETE

figure 3.55. Tableau des opérations d'accès aux fichiers.

- le fichier d'accès;
- la clé d'accès utilisée;
- l'interface, c'est-à-dire la description de l'article-courant transmis au module appelant;

L'organisation du fichier sera seulement déterminée lors de la réalisation physique des programmes (1).

A titre d'exemple, les modules d'entrée-sortie de la phase 1 du cas PETITPAS peuvent être spécifiés de la manière suivante :

```
MODULE E-S lecture-carte;
  opération d'accès READ
  fichier d'accès   carte
  interface         art-carte

MODULE E-S accès-cli;
  opération d'accès OBTAIN INSERT REPLACE
  fichier d'accès   cli
  clé d'accès       num-cli nom-cli
  interface         art-cli

MODULE E-S accès-pro;
  opération d'accès OBTAIN
  fichier d'accès   pro
  clé d'accès       num-pro lib-pro
  interface         art-pro
```

(1) L'indépendance de la structure des programmes par rapport à l'organisation des fichiers est expliquée dans [Clarinval.c].

```
MODULE E-S écriture-ligcmd;
  opération d'accès WRITE
  fichier d'accès   ligcmd
  interface         art-ligcmd

MODULE E-S écriture-justificatif;
  opération d'accès WRITE
  fichier d'accès   justificatif
  interface         art-justificatif
```

CHAPITRE IV - DEMARCHE D'ANALYSE D'UN PROBLEME SANS FLUX D'INFORMATION.

4.0. INTRODUCTION.

Dans ce chapitre, nous proposons une démarche complète d'analyse d'un problème sans flux d'information (§ 1.3.2.1.).
Nous procédons, tout comme pour les problèmes avec flux d'information, en 2 étapes :
- l'étape d'analyse conceptuelle, et
- l'étape d'analyse organique.

Nous présenterons ces 2 étapes successivement, en les illustrant par le problème de gestion des anciens de l'Institut.

4.1. PRESENTATION DU PROBLEME DE GESTION DES ANCIENS DE L'INSTITUT.

Il s'agit d'un problème de gestion du fichier des anciens étudiants de l'Institut d'Informatique de Namur.
L'énoncé du problème peut se poser ainsi (1) :
"On se propose d'automatiser la gestion du fichier des anciens de l'Institut.
Les informations (2) que l'on désire voir figurer dans ce fichier sont :
- les renseignements personnels (nom, prénom, adresse, ...), et
- les renseignements professionnels (employeur, fonction, ...).

Les opérations que l'on se propose de faire sur ce fichier sont :
- les opérations classiques de consultation et mise à jour, telles que créer un enregistrement,
  modifier un enregistrement,
  supprimer un enregistrement,
  consulter un enregistrement, et

(1) L'énoncé complet du problème est donné dans l'annexe 3.
(2) La fiche signalétique du fichier ANCIEN est dans l'annexe 4.

- la production de divers états statistiques, tels que des étiquettes,
  une liste alphabétique,
  un tableau récapitulatif.

4.2. ANALYSE CONCEPTUELLE.

4.2.1. EXPOSE DE LA DEMARCHE D'ANALYSE CONCEPTUELLE.

Nous avons défini un problème sans flux d'information comme une structure passive d'informations sur laquelle agissent des fonctions (§ 1.3.1.2.).
Dans un premier temps, nous identifierons et spécifierons la structure d'informations, représentant la partie passive du problème. Dans un second temps, nous identifierons et spécifierons la partie active du problème, c'est-à-dire les fonctions agissant sur la structure d'informations, ainsi que les messages.

4.2.2. APPLICATION DE LA DEMARCHE D'ANALYSE CONCEPTUELLE AU PROBLEME DE GESTION DES ANCIENS DE L'INSTITUT.

4.2.2.1. Identification et spécification de la structure d'informations.

1° Identification.

Dans l'énoncé du problème, nous avons une description de la fiche signalétique d'un ancien. De cette description, on déduit immédiatement le schéma conceptuel représentant les informations stockées dans la mémoire. Ce schéma conceptuel, sous forme du modèle Entité-Association est constitué de la seule entité ANCIEN dont l'expression graphique est donnée à la figure 4.1.

figure 4.1. Schéma conceptuel du problème de gestion des anciens de l'Institut.

2° Spécification.

La spécification des informations va se faire au moyen du langage D.S.L.. Pour l'entité ANCIEN, elle sera la suivante :

```
DEFINE ENTITY Ancien;
   DESCRIPTION;
      une occurrence de cette entité représente une
      personne physique ayant obtenu le diplôme de
      licencié et maître en informatique à l'Insti-
      tut d'Informatique des F.U.N.D.P.;
   IDENTIFIED BY Ident-ancien;
   CONSISTS OF Nom-ancien;
   CONSISTS OF Prénom-ancien;
   CONSISTS OF Sexe-ancien;
   CONSISTS OF Etat-civil-ancien;
   CONSISTS OF Année-diplôme-ancien;
   CONSISTS OF Orientation-ancien;
   CONSISTS OF Adresse-ancien;
   CONSISTS OF Téléphone-ancien;
   CONSISTS OF Fonction-ancien;
   CONSISTS OF Employeur-ancien;
   CONSISTS OF Adresse-employeur-ancien;

DEFINE GROUP Ident-ancien;
   DESCRIPTION;
      ce groupe identifie un ancien en spécifiant
      son année d'obtention du diplôme, son orien-
      tation, son nom et son prénom;
   IDENTIFIES Ancien;
   CONSISTS OF Année-diplôme-ancien;
   CONSISTS OF Orientation-ancien;
   CONSISTS OF Nom-ancien;
   CONSISTS OF Prénom-ancien;
```

De plus, chaque attribut appartenant à l'entité ANCIEN doit être spécifié afin de déterminer les règles syntaxiques et sémantiques de la structure d'informations. A titre d'exemple, la spécification D.S.L. de l'attribut NOM-ANCIEN est la suivante :

```
DEFINE ELEMENT Nom-ancien;
   DESCRIPTION;
      nom sous lequel l'ancien est connu de
      l'Institut d'Informatique;
   FORMAT IS X(30);
```

4.2.2.2. Identification et spécification des fonctions et des messages.

1° Identification.

Dans l'énoncé du problème, nous avons une liste d'opérations à effectuer sur la structure d'informations. A partir de cette liste, et au moyen des règles d'identification des fonctions exposées au § 1.2.2.1. nous obtenons les 7 fonctions suivantes que l'on peut répartir en 2 classes :

- les fonctions de mise à jour :
  - ajouter ‹ancien›
  - modifier ‹ancien›
  - supprimer ‹ancien›
- les fonctions de listage au sens large :
  - afficher ‹ancien›
  - éditer ‹étiquettes-ancien›
  - éditer ‹liste-alphabétique›
  - éditer ‹tableau-récapitulatif›

Ces fonctions sont indépendantes et doivent être déclenchées explicitement. Elles ont donc chacune un message d'entrée (1). On identifie ainsi pour chaque fonction son message d'entrée :

- ancien-à-ajouter
- ancien-à-modifier
- ancien-à-supprimer

(1) Chaque message d'entrée est un message externe, c'est-à-dire un message introduit par un utilisateur et donc suspect. Il devra dès lors faire l'objet d'une validation. Cette validation est intrinsèque à la spécification de la structure d'informations.

sélection-ancien-à-afficher
sélection-étiquettes-ancien
sélection-liste-alphabétique
sélection-tableau-récapitulatif

Les fonctions de listage ont pour objectif la production d'un message de sortie que nous pouvons identifier :

ancien-affiché
étiquettes-ancien
liste-alphabétique
tableau-récapitulatif

2° Spécification.

Pour spécifier ces fonctions et ces messages, nous allons utiliser le modèle statique des traitements de [Bodart-Pigneur] représenté à la figure 4.2.

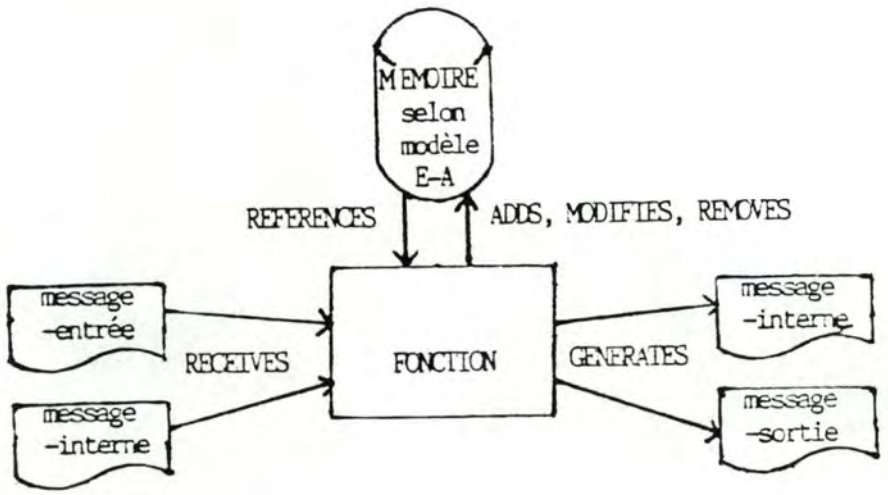


figure 4.2. Modèle statique des traitements.

Ce schéma met en évidence 3 concepts à spécifier :
- la fonction,
- les messages d'entrée et de sortie et
- la mémoire que nous avons déjà spécifiée au § 4.2.2.1.2°..

Les 2 classes de fonctions que nous avons identifiées se différencient dans le modèle statique par le fait que :

- les fonctions de mise à jour effectuent une opération d'entrée dans la structure d'informations (ajouter, modifier, supprimer);

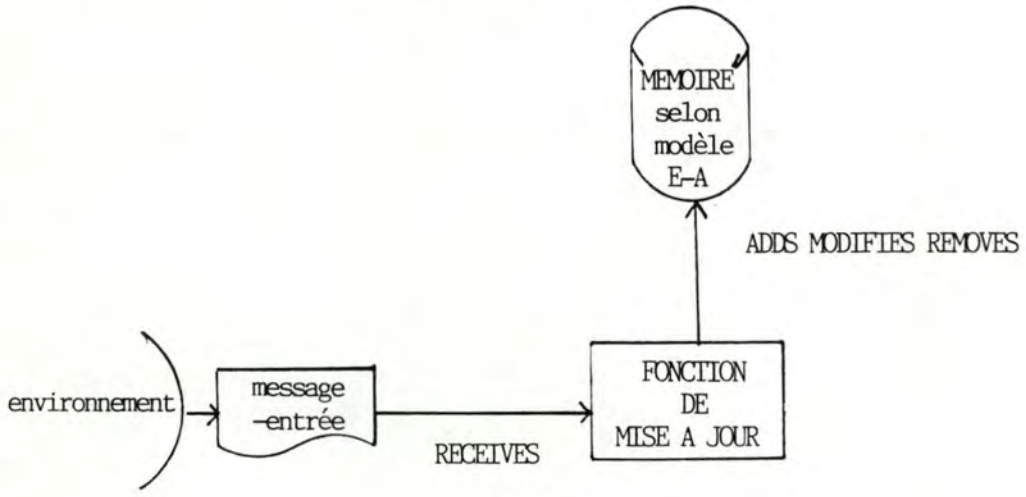


figure 4.3 Modèle statique d'une fonction de mise à jour.

- les fonctions de listage au sens large effectuent une opération de sortie de la structure d'informations (afficher, éditer)

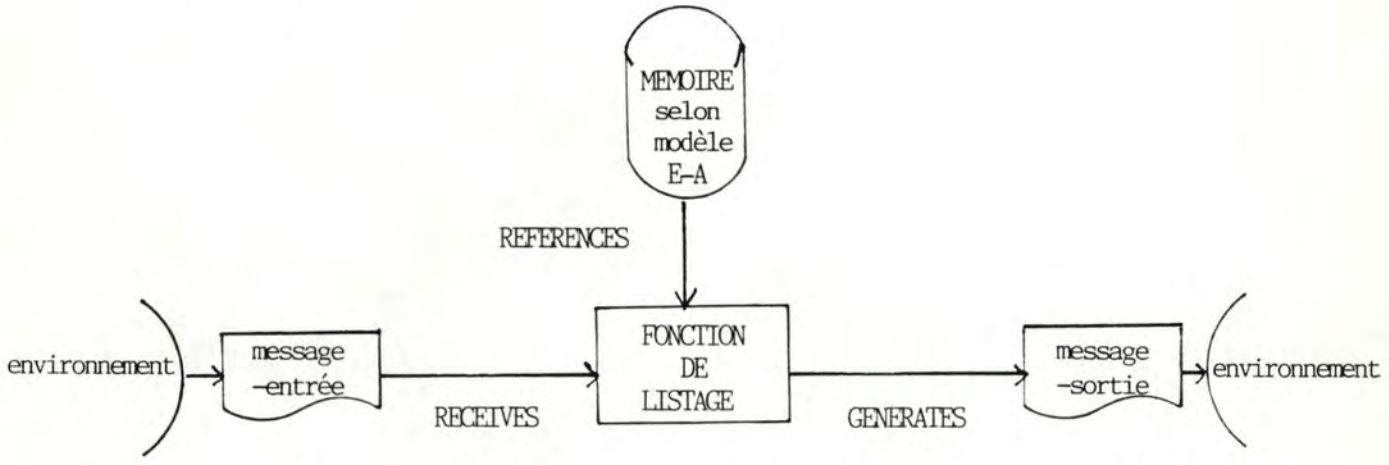


figure 4.4 Modèle statique d'une fonction de listage.

Ayant modélisé les 2 classes de fonctions, leurs spécifications peuvent être standardisées de la manière suivante :

- pour les fonctions de mise à jour

```
DEFINE PROCESS nom-fonction;
   DESCRIPTION;
      ... ;
   RECEIVES message-demande-mise-à-jour;
   ADDS
   MODIFIES mémoire;
   REMOVES
```

Le message d'entrée de ces fonctions contient 2 types d'informations :
- l'identification de l'opération demandée (ajouter, modifier, supprimer),
- l'image nouvelle de l'entité, de l'association ou des attributs à placer dans la mémoire (1).

La spécification de ce message est implicite, dérivable des spécifications de la structure d'informations.

- pour les fonctions de listage au sens large

```
DEFINE PROCESS nom-fonction;
   DESCRIPTION;
      extraction et règles de calcul éventuelles;
   RECEIVES message-demande-liste;
   GENERATES liste;
   REFERENCES mémoire;
```

Le message d'entrée de ces fonctions exprime une fonction de sélection sur la mémoire. Sa spécification consistera en la syntaxe des opérations de sélection autorisées. Il y a nécessairement un message de sortie contenant les informations extraites ou dérivées de la mémoire.

A titre d'exemple, nous spécifierons la fonction AJOUTER ‹ancien›, fonction de mise à jour et la fonction EDITER ‹tableau-récapitulatif›, fonction de listage. Nous spécifierons également les messages associés à ces fonctions : ANCIEN-A-AJOUTER, SELECTION-TABLEAU-RECAPITULATIF, TABLEAU-RECAPITULATIF. Ces spécifications se feront en D.S.L. de la manière suivante :

---

(1) Pour la fonction de suppression, l'image est nulle.

```
DEFINE PROCESS Ajouter-ancien;
   DESCRIPTION;
      ajouter "Ancien-à-ajouter" dans la collection
      d'anciens;
   RECEIVES Ancien-à-ajouter;
   ADDS Ancien;

DEFINE MESSAGE Ancien-à-ajouter;
   DESCRIPTION;
      ce message représente un ancien à ajouter
      à la collection des anciens;

   CONSISTS OF Nom-ancien;
   CONSISTS OF Prénom-ancien;
   CONSISTS OF Sexe-ancien;
   CONSISTS OF Etat-civil-ancien;
   CONSISTS OF Année-diplôme-ancien;
   CONSISTS OF Orientation-ancien;
   CONSISTS OF Adresse-ancien;
   CONSISTS OF Téléphone-ancien;
   CONSISTS OF Fonction-ancien;
   CONSISTS OF Employeur-ancien;
   CONSISTS OF Adresse-employeur-ancien;

DEFINE PROCESS Editer-tableau-récapitulatif;
   DESCRIPTION;
      - extraire de la collection d'anciens, les anciens
        de l' (des) année(s) et orientation(s) sélectionnées
        par "Sélection-tableau-récapitulatif"
      - en calculer le nombre par année-orientation, et
        totaliser par année, par orientation et globalement;
   RECEIVES Sélection-tableau-récapitulatif;
   GENERATES Tableau-récapitulatif;

DEFINE MESSAGE Sélection-tableau-récapitulatif;
   DESCRIPTION;
      ce message représente une sélection d'année(s) et
      d'orientation(s) pour un tableau récapitulatif;

   CONSISTS OF Année-début;
   CONSISTS OF Année-fin;
   CONSISTS OF Orientations-choisies;

DEFINE MESSAGE Tableau-récapitulatif;
   DESCRIPTION;
      tableau récapitulatif édité pour les anciens sélectionnés;
   LAYOUT;
```

orientation / année	GESTION	SCIENTIFIQUE	TOTAL
1973	..	..	...
1974	..	..	...
1975	..	..	...
TOTAL	...	...	....

```
   ;
   CONSISTS OF Année-sélectionnée;
   CONSISTS OF Nombre-anciens-gestion-par-année;
   CONSISTS OF Nombre-anciens-scientifique-par-année;
   CONSISTS OF Nombre-anciens-total-par-année;
   CONSISTS OF Nombre-anciens-total-gestion;
   CONSISTS OF Nombre-anciens-total-scientifique;
   CONSISTS OF Nombre-anciens-total;
```

4.3. ANALYSE ORGANIQUE.

4.3.1. EXPOSE DE LA DEMARCHE D'ANALYSE ORGANIQUE.

L'analyse organique se fera en 2 temps, tout comme l'analyse conceptuelle. Dans un premier temps, nous procéderons à la construction de l'architecture des données par réinterprétation du modèle des informations de l'analyse conceptuelle. Cette étape constitue l'analyse de la partie passive du problème. Dans un second temps, nous construirons l'architecture des traitements par réinterprétation du modèle des traitements de l'analyse conceptuelle. Cette étape constitue l'analyse de la partie active du problème.

4.3.2. APPLICATION DE LA DEMARCHE D'ANALYSE ORGANIQUE AU PROBLEME DE GESTION DES ANCIENS DE L'INSTITUT.

4.3.2.1. Architecture des données.

1° Transformation des informations en fichiers.

La mémoire, représentant la structure d'informations, se transforme en fichiers logiques qui seront concrétisés par des fichiers physiques. Ces transformations obéissent aux mêmes règles que celles exposées pour un problème avec flux d'information au § 3.3.2.1..
Quant aux messages d'entrée et de sortie des fonctions, ils sont transformés en fichiers virtuels respectivement de statut M et N.

2° Spécification des fichiers.

La spécification des fichiers virtuels et réels se fera de manière identique à celle exposée pour les problèmes avec flux d'information au § 3.3.2.1.4..

4.3.2.2. Architecture des traitements.

1° Transformation des fonctions en couplage de modules.

L'analyse conceptuelle a identifié les fonctions centrales du problème. Les fonctions auxiliaires de validation et de mise en page n'ont pas été mises en évidence, car elles sont implicites dans la spécification des objets en définissant des contraintes de validité sur la mémoire et en décrivant les messages d'entrée et de sortie. La spécification de ces fonctions aurait constitué une redondance dans l'analyse conceptuelle.

Aux fonctions centrales correspondent des modules fonctionnels. Ainsi, pour le problème de la gestion des anciens de l'Institut, on a les correspondances suivantes :

ajouter <ancien>	ADDANC
modifier <ancien>	MODANC
supprimer <ancien>	DELANC
afficher <ancien>	AFFANC
éditer <étiquettes-ancien>	EDTETI
éditer <liste-alphabétique>	EDTLST
éditer <tableau-récapitulatif>	EDTTAB

Aux fonctions auxiliaires correspondent des modules que nous qualifierons d'auxiliaires fonctionnels. Ainsi, on a les correspondances suivantes :

valider <ancien-à-ajouter>	VALADD
valider <ancien-à-modifier>	VALMOD
valider <ancient-à-supprimer>	VALDEL
valider <sélection-ancien-à-afficher>	VALAFF
valider <sélection-étiquettes-ancien>	VALETI
valider <sélection-liste-alphabétique>	VALLST
valider <sélection-tableau-récapitulatif>	VALTAB
mise-en-page <étiquettes-ancien>	MPGETI
mise-en-page <liste-alphabétique>	MPGLST
mise-en-page <tableau-récapitulatif>	MPGREC

S'ajoutent à ces 2 types de modules, les modules d'entrée-sortie réalisant l'accès aux informations.

En couplant ces différents modules, on obtient donc un flux au niveau de l'analyse organique. Le schéma du flux diffère selon la classe de fonctions :

- pour les fonctions de mise à jour, le flux peut être présenté sous forme standard tel qu'à la figure 4.5.;
- pour les fonctions de listage au sens large, le flux peut être représenté sous forme standard tel qu'à la figure 4.6..

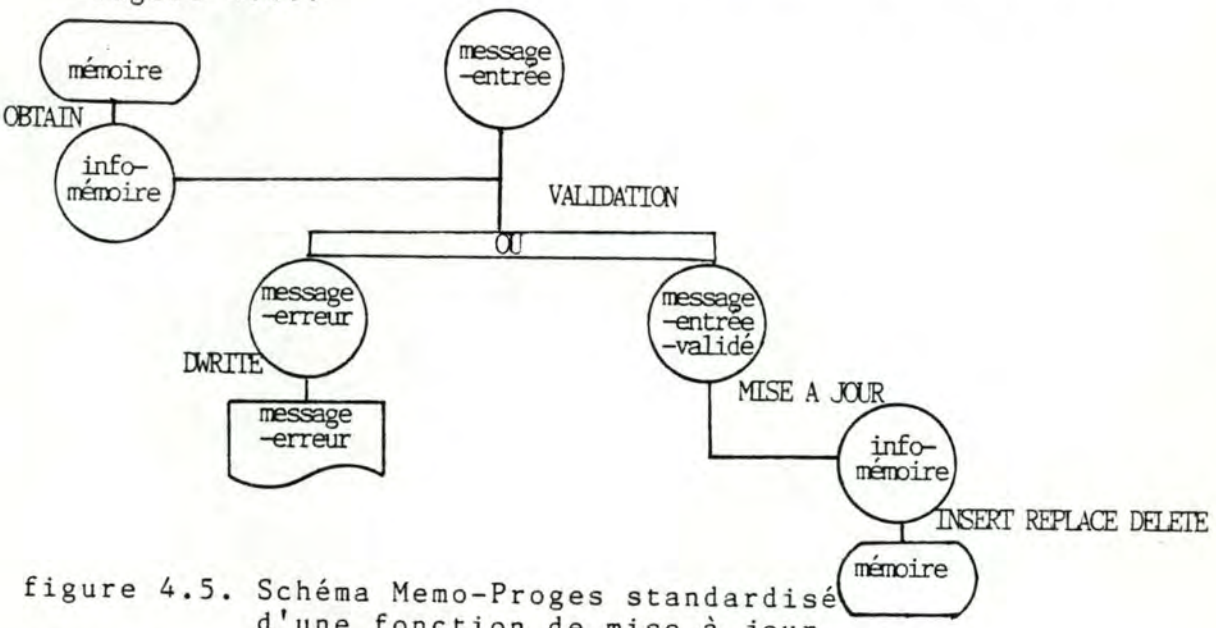


figure 4.5. Schéma Memo-Proges standardisé d'une fonction de mise à jour.

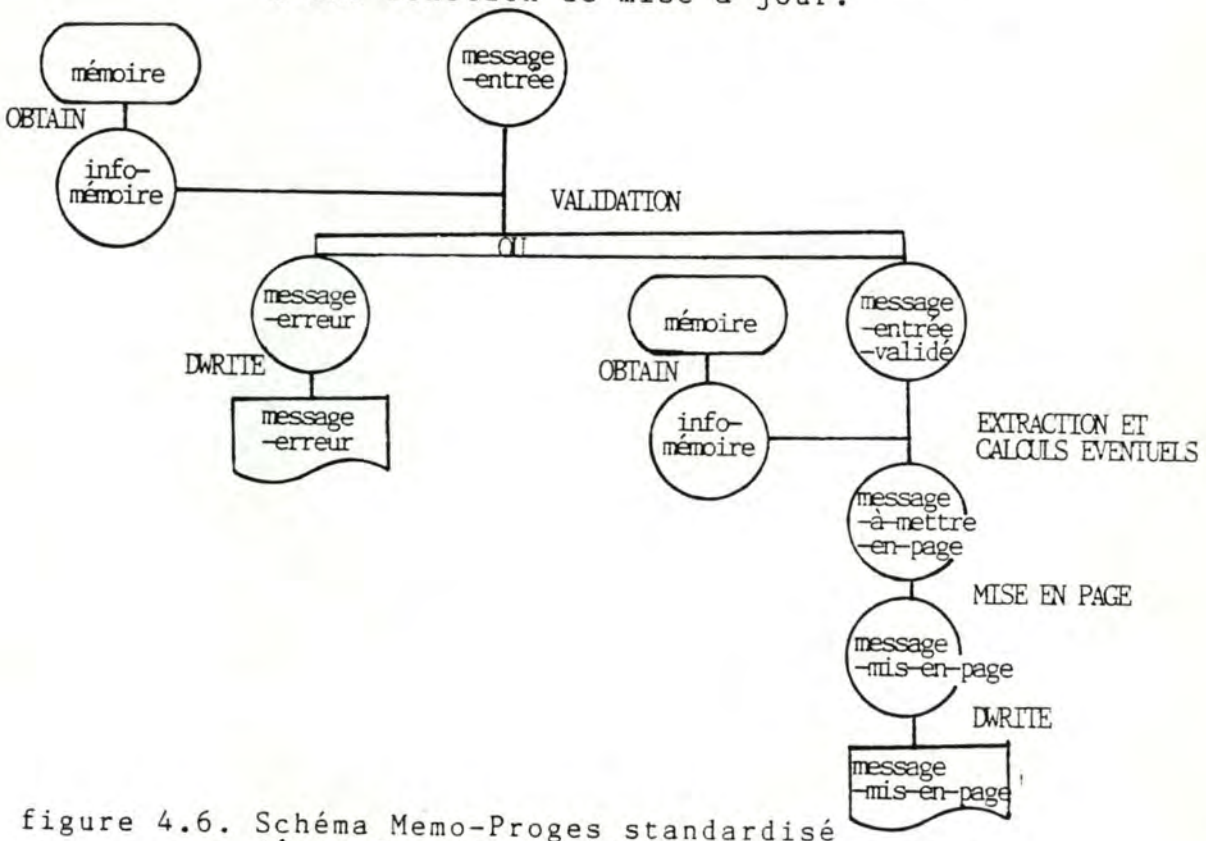


figure 4.6. Schéma Memo-Proges standardisé d'une fonction de listage au sens large.

Ces 2 schémas mettent en évidence l'apparition de messages d'erreur résultant de la fonction de validation. Ces messages seront spécifiés par l'analyste organique.

2° Transformation du catalogue en module coordinateur général.

Lors de l'analyse conceptuelle, nous avons modélisé un problème sans flux d'information sous forme d'un catalogue de fonctions. L'introduction d'un module coordinateur général concrétise ce catalogue en aiguillant l'exécution vers la fonction à réaliser grâce au message d'entrée. Ceci est représenté à la figure 4.7.

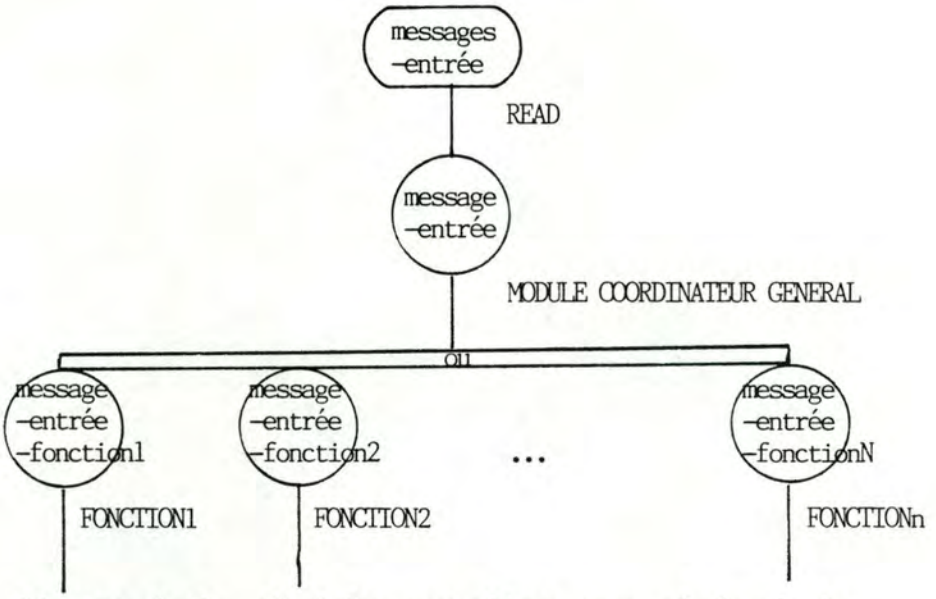


figure 4.7. Module coordinateur général du catalogue des fonctions.

3° Spécification des modules.

A. Modules fonctionnels.
La spécification des modules fonctionnels se fera de manière identique pour les problèmes avec ou sans flux d'information (§ 3.3.2.2.3.A).

B. Modules auxiliaires.
Les modules auxiliaires se répartissent en 2 classes :
- les modules de validation et
- les modules de mise en page.

Les modules de validation effectuent plusieurs types de validation :

- la validation syntaxique pour laquelle les contraintes de validité sont exprimées dans le format des éléments spécifiés en D.S.L. lors de l'analyse conceptuelle;
- la validation sémantique (contrôle d'évolution, contrôle inter-données, ...) pour laquelle les contraintes de validité sont exprimées dans la description des entités, associations et éléments spécifiés en D.S.L. lors de l'analyse conceptuelle.

A chaque contrainte devra être associé un message d'erreur dont la spécification revient à l'analyste organique.
Les modules de mise en page seront spécifiés à partir de la description des messages de sortie faite en D.S.L. lors de l'analyse conceptuelle.

C. Modules d'entrée-sortie.

Les modules d'entrée-sortie seront spécifiés de manière identique aux modules d'entrée-sortie des problèmes avec flux d'information (§ 3.3.2.2.3.C).

D. Module coordinateur général.

La spécification standardisée du module coordinateur général MCG est la suivante :

argument d'entrée : message-entrée comprenant entre autres une information permettant de sélectionner la fonction à exécuter.

fontion : aiguille le message-entrée vers la fonction I à exécuter

argument de sortie : message-entrée-fontionI

CHAPITRE V - IMPLICATIONS DU MODE D'EXPLOITATION SUR L'ARCHITECTURE DES PROGRAMMES.

5.0. INTRODUCTION.

Ce chapitre a pour but de situer les implications dues au choix d'un mode d'exploitation des programmes. Nous verrons en quoi le mode batch et le mode interactif influencent l'architecture des programmes. Quant au mode transactionnel, il s'agit d'un problème qui est du ressort du système d'exploitation (1).

5.1. MODE BATCH ET MODE INTERACTIF.

Ce paragraphe a pour objectif de décrire les différences sur la programmation qu'impliquent le mode batch et le mode interactif. Ces différences se situent à 2 niveaux :

- au niveau de l'architecture :
  - forme des fonctions d'acquisition,
  - couplage des modules réalisant ces fonctions, et
  - place du module MAIN dans l'architecture;
- au niveau de la conception des fonctions.

Nous commencerons par définir le contexte dans lequel nous travaillons, en caractérisant les modes batch et interactif.

5.1.1. Définition du contexte d'exploitation.

L'interactivité est caractérisée par le fait que deux processus simultanés échangent entre eux des messages. Le type d'interactivité que nous avons pris en considération, dans le cadre de notre travail, est l'interactivité

(1) Les définitions des modes d'exploitation et de tous les concepts associés sont tirées de [Clarinval.b].

entre un processus programmé, le programme en cours d'exécution, et un processus terminal, l'utilisateur à sa console.
Le mode batch et le mode interactif se différencient par le fait que le mode interactif permet ce type d'interactivité, tandis que le mode batch ne le permet pas.
De plus, nous supposons, que dans le mode interactif, le mode de dialogue entre le processus programmé et le processus terminal se fait selon le mode conversationnel.
On dit que 2 processus P1 et P2 dialoguent en mode conversationnel, quand P1 ayant envoyé un message à P2, il n'est pas autorisé à envoyer un second message avant que P2 ait lui-même envoyé un message de réponse à P1.
Ce mode de dialogue est largement utilisé, car il a pour avantage d'offrir une guidance à l'utilisateur.

5.1.2. Comparaison des architectures des programmes en mode batch et en mode interactif.

A. Différences au niveau de l'architecture.

1° Forme des fonctions d'acquisition.

Les fonctions dont la forme peut changer suivant le mode d'exploitation dans lequel on travaille sont les fonctions d'acquisition d'informations. Il s'agit des fonctions qui, dans le modèle conceptuel, transforment un message initial en message validé. A titre d'exemple, pour la phase "ENREGISTREMENT-BON-DE-COMMANDE-CLIENT", ce sont les fonctions :

- AFFICHER ‹résultat-contrôle-identité›,
- AFFICHER ‹résultat-contrôle-ligne›,
- AFFICHER ‹résultat-contrôle-total›.

Ces fonctions ont pour résultat de produire une structure d'informations valide et significative pour les fonctions de production de résultats. Ces fonctions de production travaillant à partir d'une structure d'informations valide, elles seront indépendantes du mode d'exploitation utilisé.

En Memo-Proges, les modules d'acquisition doivent fournir aux modules de production un enregistrement valide, unité de traitement de Memo-Proges. La différence entre les modes batch et interactif est due à l'unité de lecture :

- en batch, l'unité de lecture est l'enregistrement. Ceci se justifie par le fait que passer au programme une structure de données construite a priori permet des économies de construction. Un programme batch itérera donc sur l'enregistrement.
- en interactif "conversationnel", l'unité de lecture est habituellement la donnée. L'acquisition par donnée se justifie par une recherche de facilité de correction. Dans un programme interactif, les modules de production itéreront sur l'enregistrement, tandis que les modules d'acquisition itéreront en plus sur les données dans l'enregistrement.

Les modèles d'acquisition en mode batch et interactif sont présentés respectivement aux figures 5.1. et 5.2..

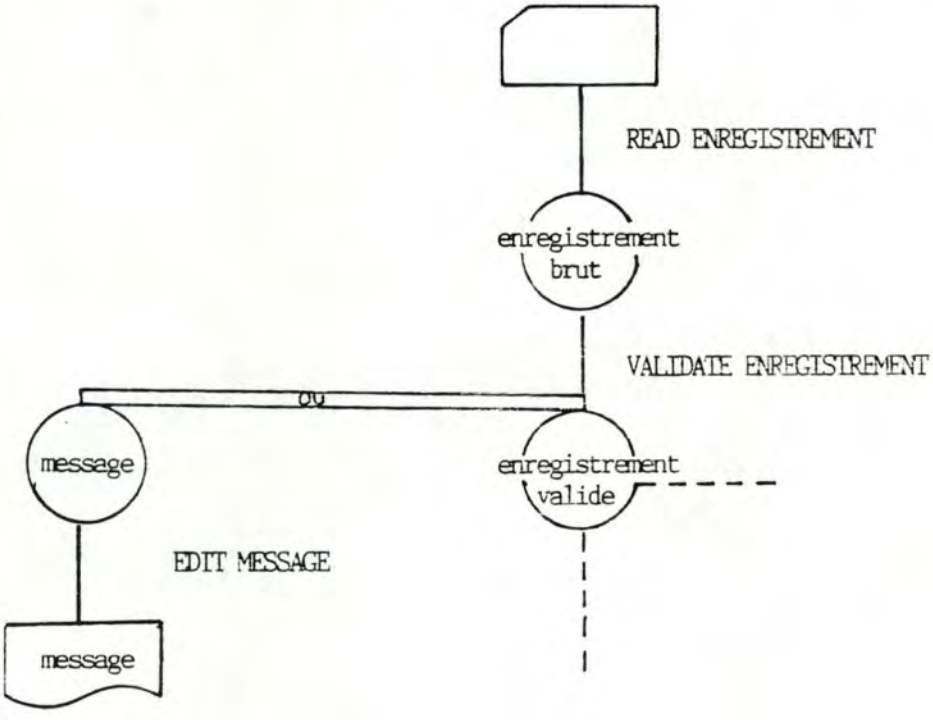


figure 5.1 . Modèle d'acquisition en mode batch.

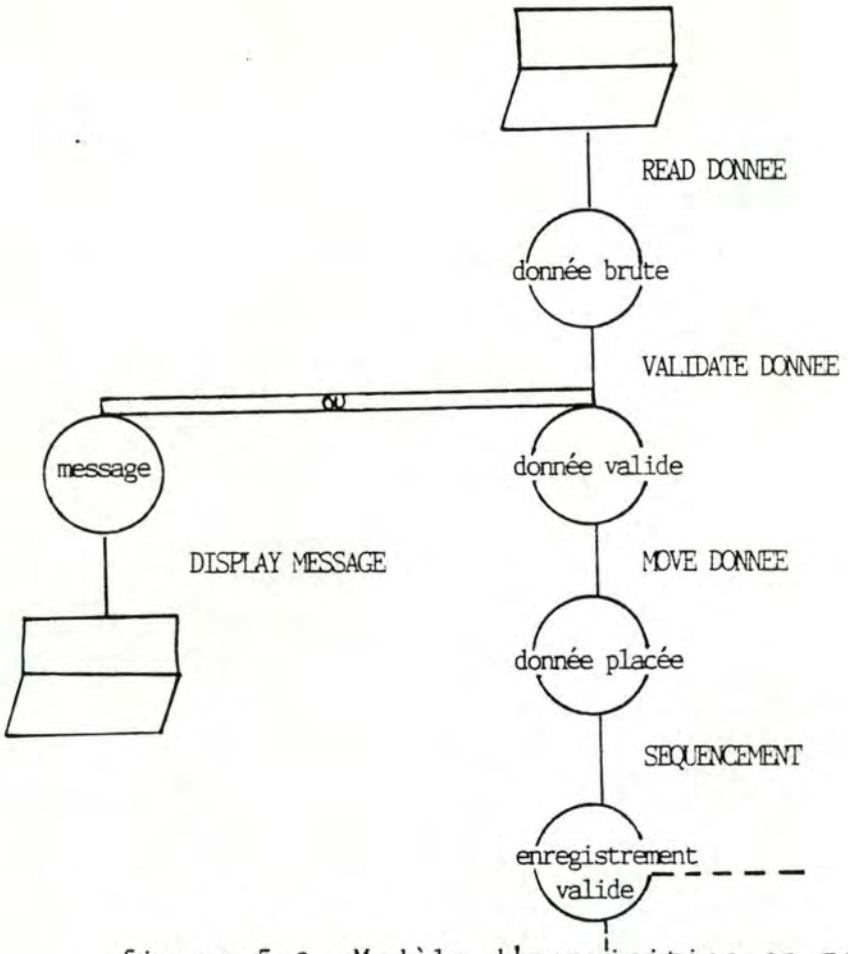
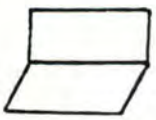


figure 5.2. Modèle d'acquisition en mode interactif.

On aura ainsi dans le modèle batch un module de lecture d'un enregistrement (READ) suivi d'un module de validation de cet enregistrement (VALIDATE).
Dans le mode interactif, on a les modules suivants :

- module de lecture d'une donnée au terminal identifiée soit par un message, soit par le placement du curseur,
- module de validation de la donnée lue,
- module de transfert de la donnée dans l'enregistrement,
- module de séquencement de la lecture qui doit indiquer l'ordre de lecture des données élémentaires composant l'enregistrement à produire. Ce module peut en outre - à titre de guidance - faire intervenir d'autres fonctions telles que affichage d'un fond d'écran ou formulaire d'encodage, affichage de textes explicatifs, affichage de valeurs initiales ou précédentes ...



représente une communication avec un utilisateur à une console.

FORM représente l'accès par gestionnaire d'écran à la console.

Pour réaliser les fonctions d'acquisition interactives en Memo-Proges, nous avons utilisé le gestionnaire d'écran FORM (1). Il est impossible, dans le cadre de ce mémoire, d'entrer en détail dans le gestionnaire d'écran. Nous en resterons aux principes essentiels. Ce gestionnaire réalise les sous-fonctions associées à une fonction d'acquisition interactive représentée à la figure 5.2 : lecture d'une donnée, validation, séquencement ...
La standardisation de ces fonctions a été opérée en associant à chaque donnée un descripteur contenant :

- la localisation dans l'enregistrement : position, longueur
- l'identification au terminal :
  - papier : message d'identification
  - écran : position et attributs de visualisation
- les conditions de validité :
  - de type syntaxique : alphabet, longueur minimum ...
  - liste des valeurs autorisées
  - ...
- le séquencement :
  - l'ordre de présentation des descripteurs définit le séquencement par défaut
- dessin et texte du fond d'écran associé
- etc.

Ces descripteurs sont regroupés dans un fichier descripteur d'écran.
L'outil effectue toutes les fonctions décrites plus haut en consultant ce fichier descripteur d'écran (2). Il peut être appelé sous trois modes :

- INPUT : affichage de valeurs initiales, puis lecture
- OUTPUT : affichage seulement
- DELETE : effacement de l'écran.

L'interface entre le programme appelant et l'outil est l'enregistrement valide attendu. Cet enregistrement est préfixé par le nom du descripteur à consulter.

---

(1) Le gestionnaire FORM a été développé par Clarinval et fonctionne au Groupe S.

(2) L'utilisateur peut programmer une sous-routine de séquencement pour introduire des variations dans le parcours du formulaire.

2° Couplage des modules réalisant les fonctions d'acquisition.

L'acquisition des données en batch et en interactif ne se fait pas de la même manière.
En mode interactif, les modules d'acquisition des données (FORM) sont appelés en indiquant l'enregistrement que le programme veut qu'ils lui transmettent. C'est le programme qui guide l'acquisition des données.
En mode batch, ceci est impossible, puisque les données sont présentes avant l'exécution du programme. Les modules d'acquisition (READ) sont appelés pour recevoir a priori n'importe quel type d'enregistrement.

Cette différence dans l'acquisition des données va donner lieu, au niveau de l'architecture, à 2 schémas.
En mode batch, il y aura après le module de lecture des enregistrements, un module d'aiguillage des enregistrements vers le traitement adéquat, qui sera en l'occurrence la validation selon le type d'enregistrement. Ce schéma est présenté à la figure 5.3.

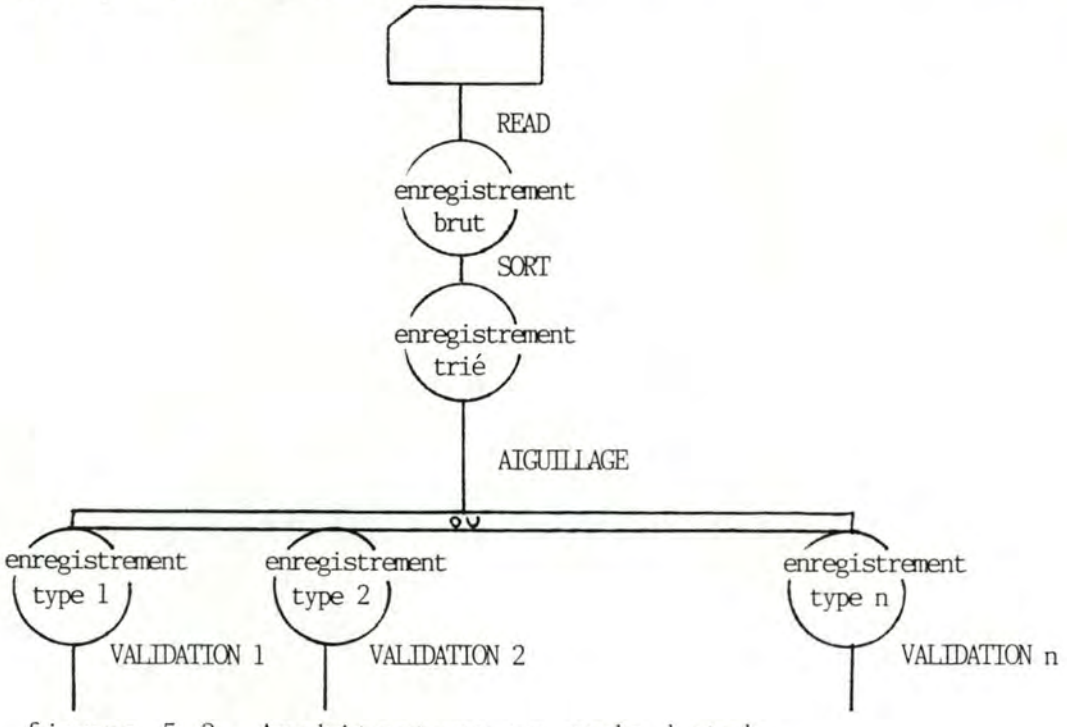


figure 5.3. Architecture en mode batch

Le module de tri SORT a été introduit pour assurer un bon ordonnancement des enregistrements.

A titre illustratif, nous présentons l'architecture d'un programme batch réalisant l'acquisition et la validation de bons de commande. Ces bons de commande sont éclatés en un enregistrement "en-tête" et 1 à n enregistrements "lignes" stockés sur le même fichier "CARD". Le schéma de l'architecture peut se présenter comme à la figure 5.4..

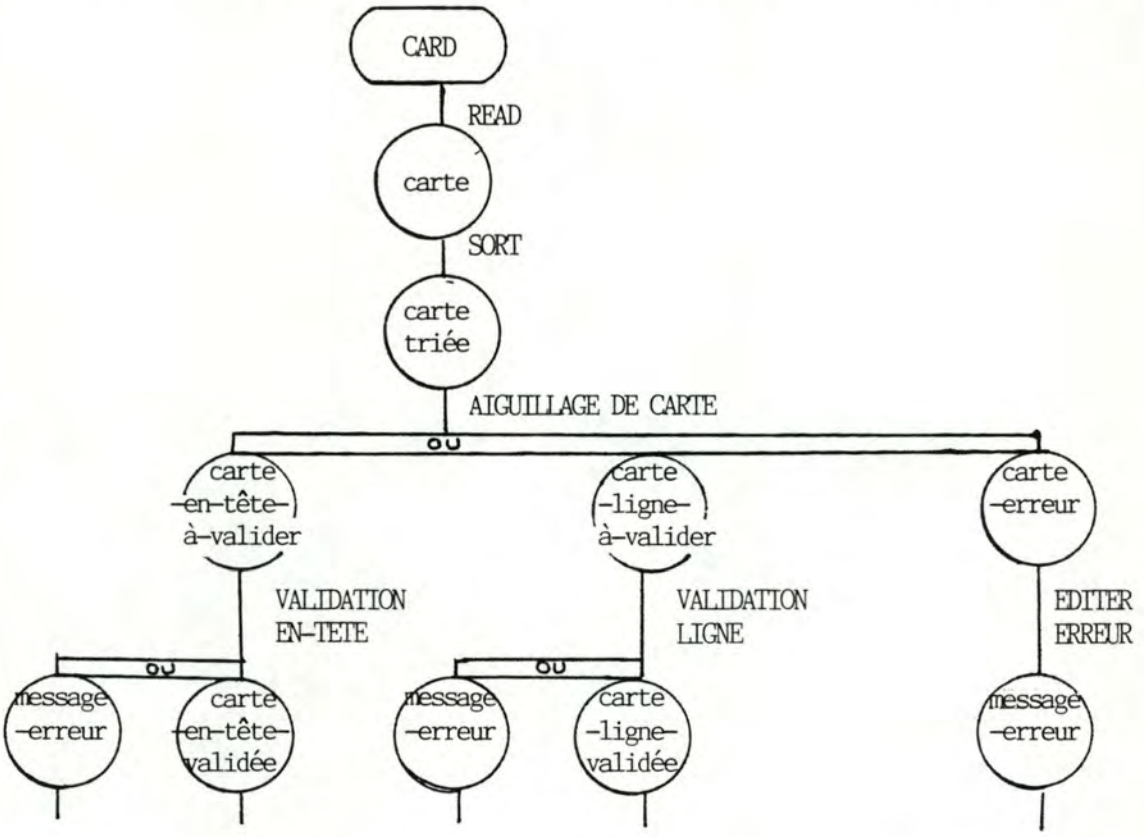


figure 5.4. Architecture de l'acquisition de bons de commande en mode batch.

En mode interactif, il faut répartir dans le programme les différents appels aux modules d'acquisition pour les différents types d'enregistrements. Ce schéma est présenté à la figure 5.5.

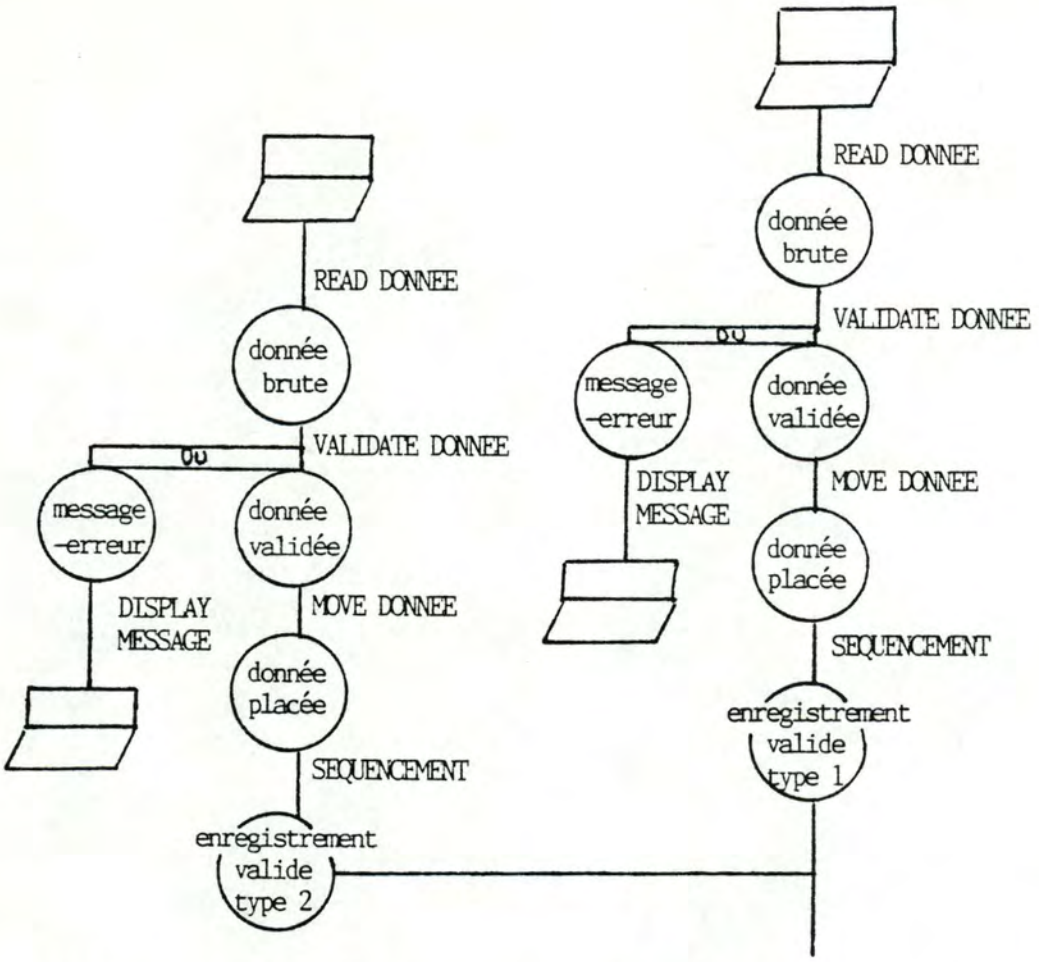


figure 5.5. Architecture en mode interactif.

A titre illustratif, nous présentons l'architecture d'un programme interactif réalisant l'acquisition et la validation de bons de commande. Ces bons de commande sont éclatés, comme dans l'exemple en mode batch, par "en-tête" et "lignes". Le schéma de l'architecture peut se présenter comme à la figure 5.6..

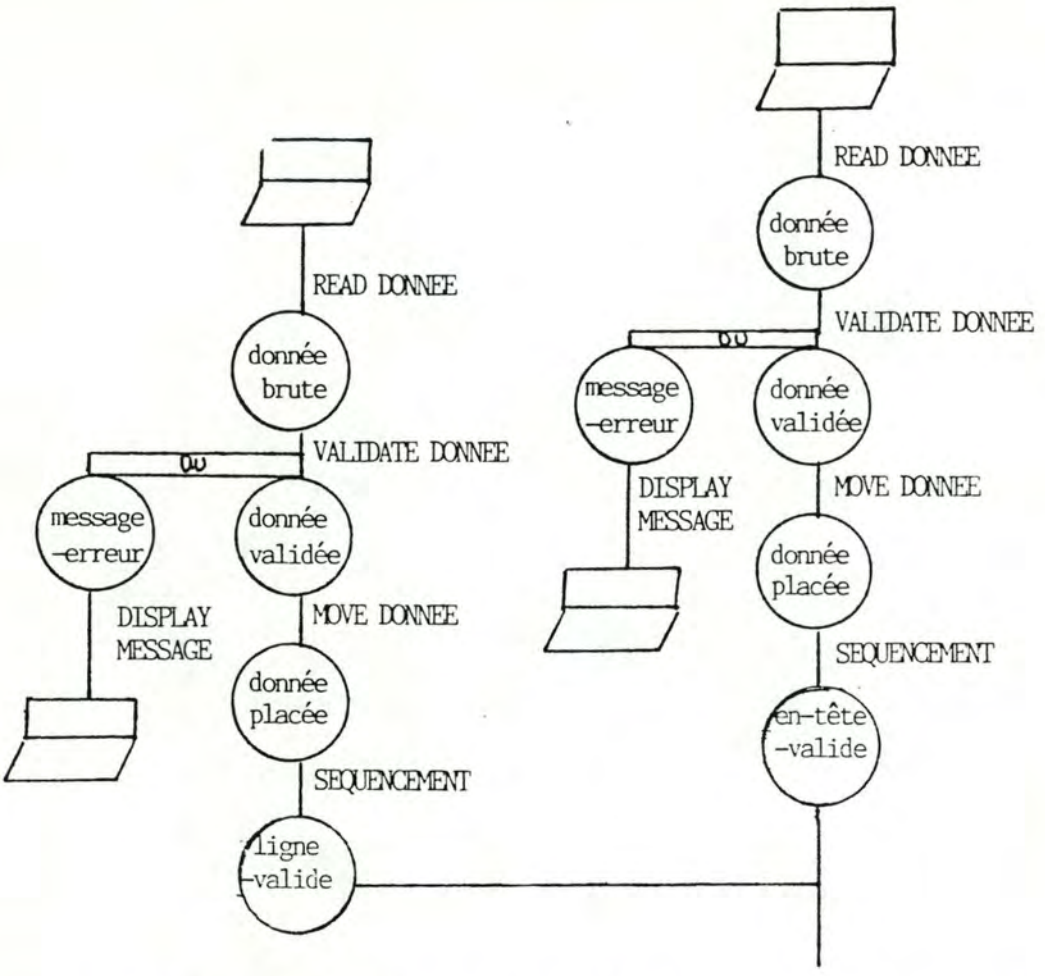


figure 5.6. Architecture de l'acquisition de bons de commande en mode interactif.

Il est intéressant de remarquer qu'on pourrait avoir, en batch, le même type de schéma que le schéma interactif, si on pouvait accéder en batch par sous-ensembles aux enregistrements de types différents. Ceci serait rendu possible si, par exemple, on faisait l'hypothèse que les enregistrements "en-tête" et les enregistrements "lignes" des bons de commande étaient stockés sur des supports différents. Cette hypothèse, bien que théorique en batch, est proche d'une structure de stockage sous forme de base de données. On aurait alors le shéma de la figure 5.7..

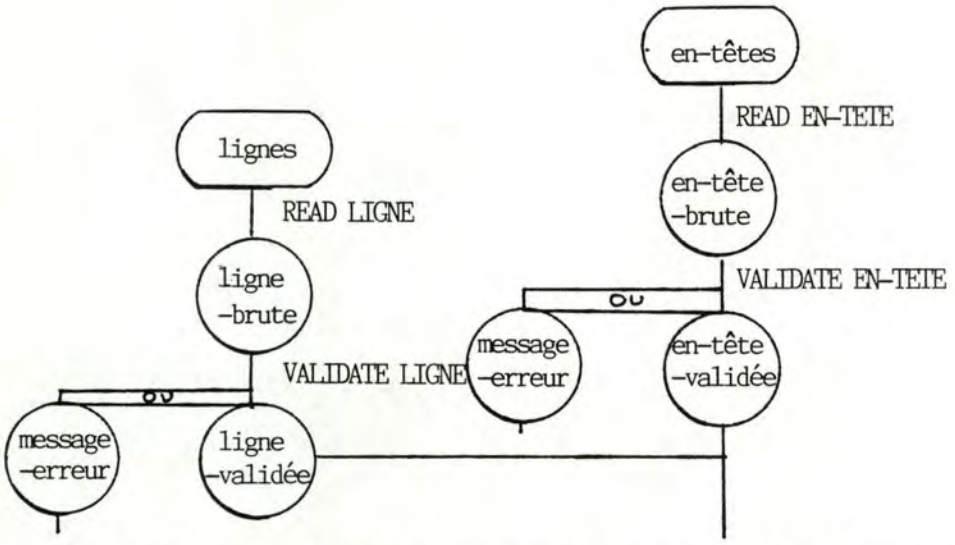


figure 5.7. Architecture théorique de l'acquisition de bons de commande batch.

On procédera d'abord à la lecture de l'en-tête, suivie de sa validation; si l'en-tête est valide, on la passe à un module suivant qui fait l'acquisition des lignes et les valide.
La différence entre les architectures ne dépend donc pas seulement du mode d'exploitation, mais également du mode de répartition des données.

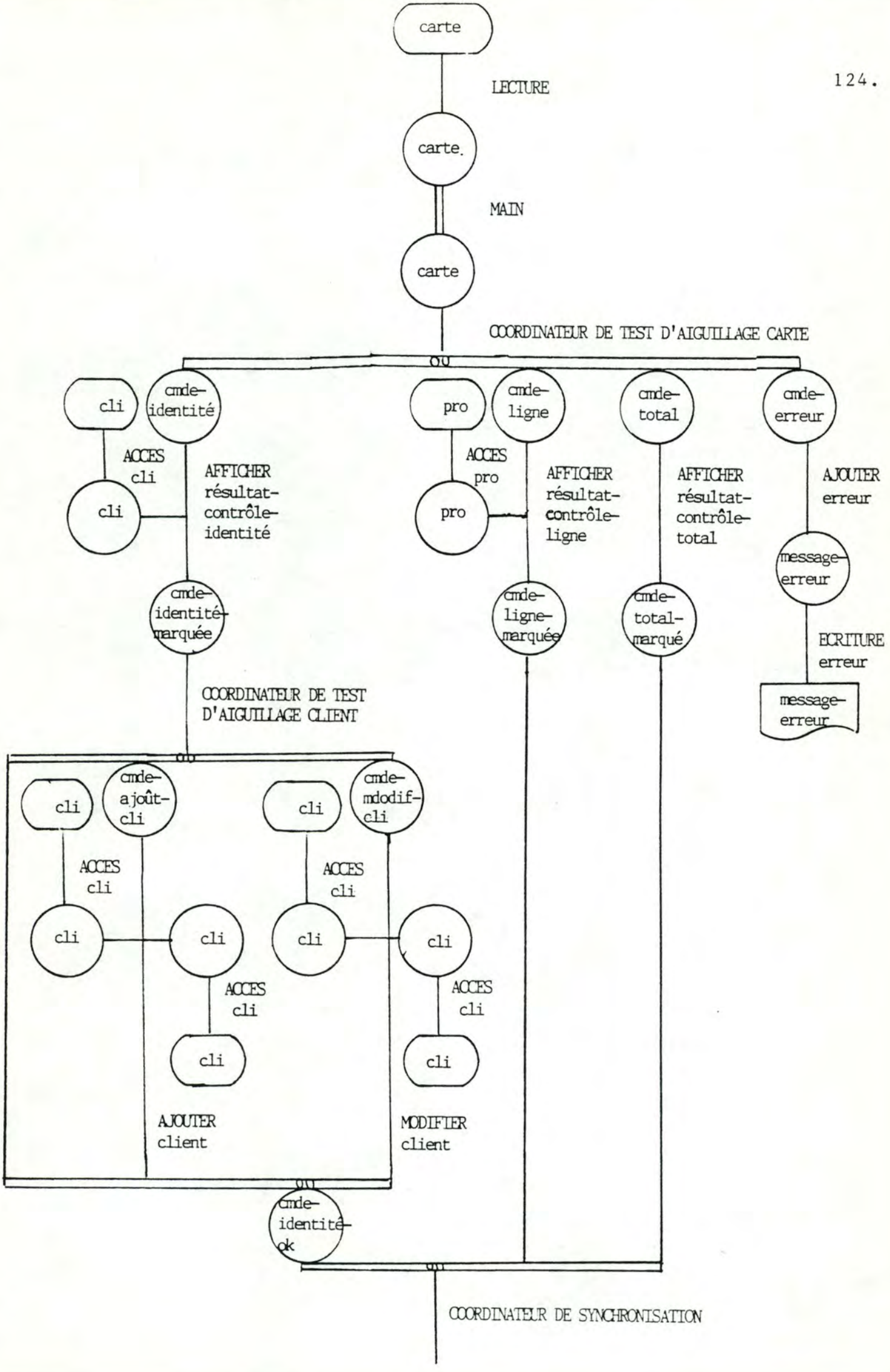
3° Place du module MAIN dans l'architecture.

La seule règle impérative qui existe en Memo-Proges, est de placer le module MAIN à un couplage N-M (couplage standard).
Dans l'architecture en mode batch, nous l'avons placé le plus près du module de lecture initial pour des raisons exposée dans Clainval.a .
Dans l'architecture en mode interactif, sa place a été contrainte par l'utilisation du gestionnaire d'écran (1).

4° Illustration.

En guise d'illustration de ces différences, nous présentons les architectures en modes batch et interactif de la phase ENREGISTREMENT-BON-DE-COMMANDE-CLIENT aux figures 5.8. et 5.9..

(1) Le gestionnaire d'écran a été prélinké sous forme d'un module unique en forme d'acquisition.


carte
LECTURE
carte.
MAIN
carte
COORDINATEUR DE TEST D'AIGUILLAGE CARTE
cli
cmde-identité
pro
cmde-ligne
cmde-total
cmde-erreur
ACCES cli
cli
AFFICHER résultat-contrôle-identité
ACCES pro
pro
AFFICHER résultat-contrôle-ligne
AFFICHER résultat-contrôle-total
AJOUTER erreur
cmde-identité-marquée
cmde-ligne-marquée
cmde-total-marqué
message-erreur
ECRITURE erreur
message-erreur
COORDINATEUR DE TEST D'AIGUILLAGE CLIENT
cli
cmde-ajoût-cli
cli
cmde-mdodif-cli
ACCES cli
cli
cli
ACCES cli
cli
cli
ACCES cli
cli
ACCES cli
cli
AJOUTER client
MODIFIER client
cmde-identité-ok
COORDINATEUR DE SYNCHRONISATION

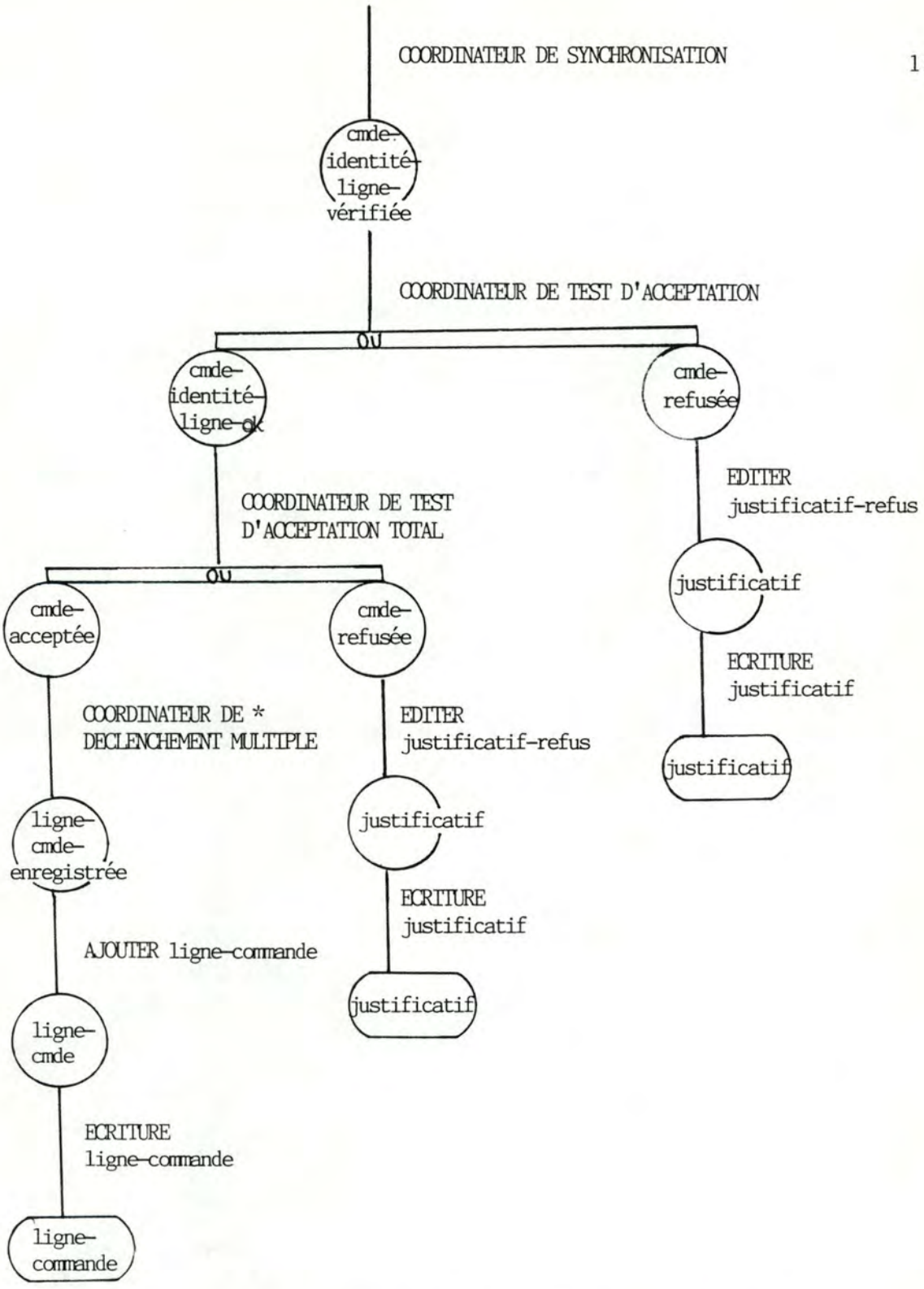
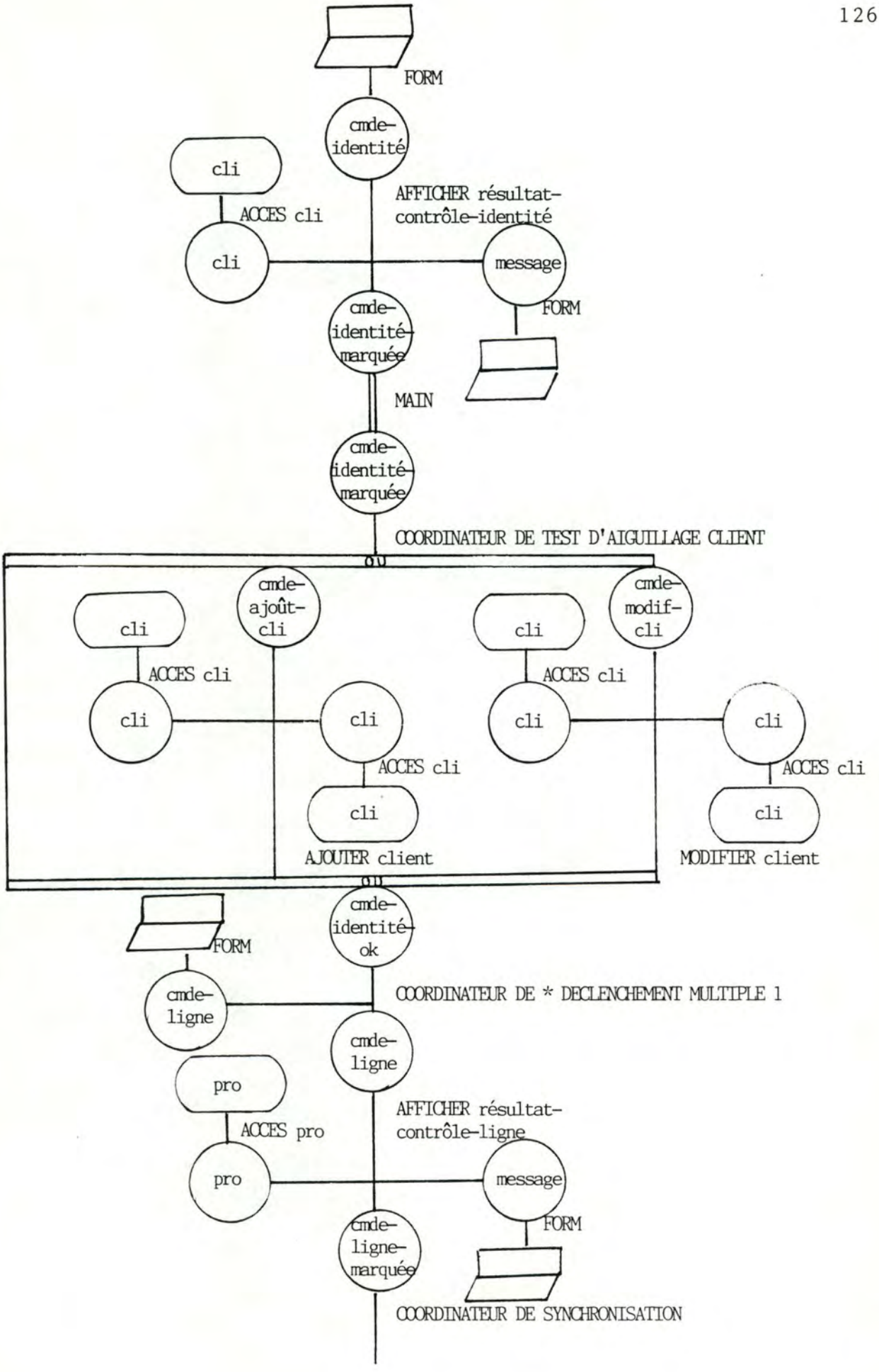


figure 5.8. Architecture de la phase ENREGISTREMENT-BON-DE-COMMANDE-CLIENT en mode batch.


FORM
cmde-identité
cli
ACCES cli
cli
AFFICHER résultat-contrôle-identité
message
FORM
cmde-identité-marquée
MAIN
cmde-identité-marquée
COORDINATEUR DE TEST D'AIGUILLAGE CLIENT
ou
cmde-ajoût-cli
cmde-modif-cli
cli
ACCES cli
cli
cli
ACCES cli
cli
cli
ACCES cli
cli
cli
ACCES cli
cli
AJOUTER client
MODIFIER client
ou
FORM
cmde-identité-ok
cmde-ligne
COORDINATEUR DE * DECLENCHEMENT MULTIPLE 1
cmde-ligne
pro
ACCES pro
pro
AFFICHER résultat-contrôle-ligne
message
FORM
cmde-ligne-marquée
COORDINATEUR DE SYNCHRONISATION

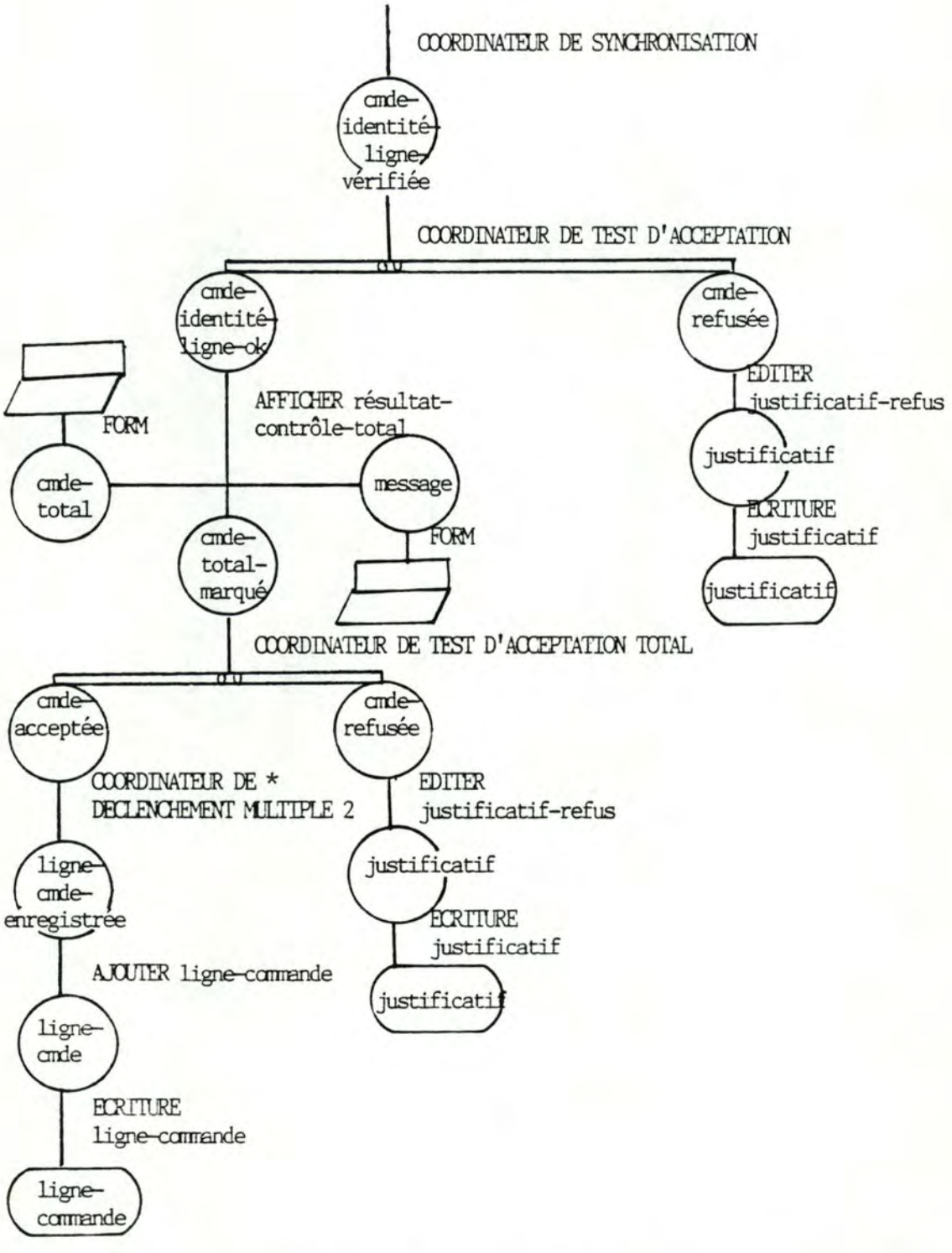


figure 5.9. Architecture de la phase ENREGISTREMENT-BON-DE-COMMANDE-CLIENT en mode interactif.

B. Différence au niveau de la conception des fonctions.

Le mode d'exploitation influence la conception de certaines fonctions. Les différences sont particulièrement sensibles dans les fonctions de mise à jour de fichier rencontrées dans les problèmes sans flux d'information.
Prenons pour exemple, la fonction MODIFIER <ancien>.
En batch, cette fonction serait conçue de la manière suivante :
On aurait un module de lecture de l'ancien à modifier - sous forme de la clé d'accès et des valeurs nouvelles à mettre à jour-, suivi d'un module de validation syntaxique et d'un module d'interclassement de ces nouvelles valeurs avec les anciennes. On obtiendrait ainsi le nouvel enregistrement proposé sur lequel on procéderait à la validation inter-données. Si cet enregistrement est correct, on met à jour le fichier. Ceci est présenté à la figure 5.10..

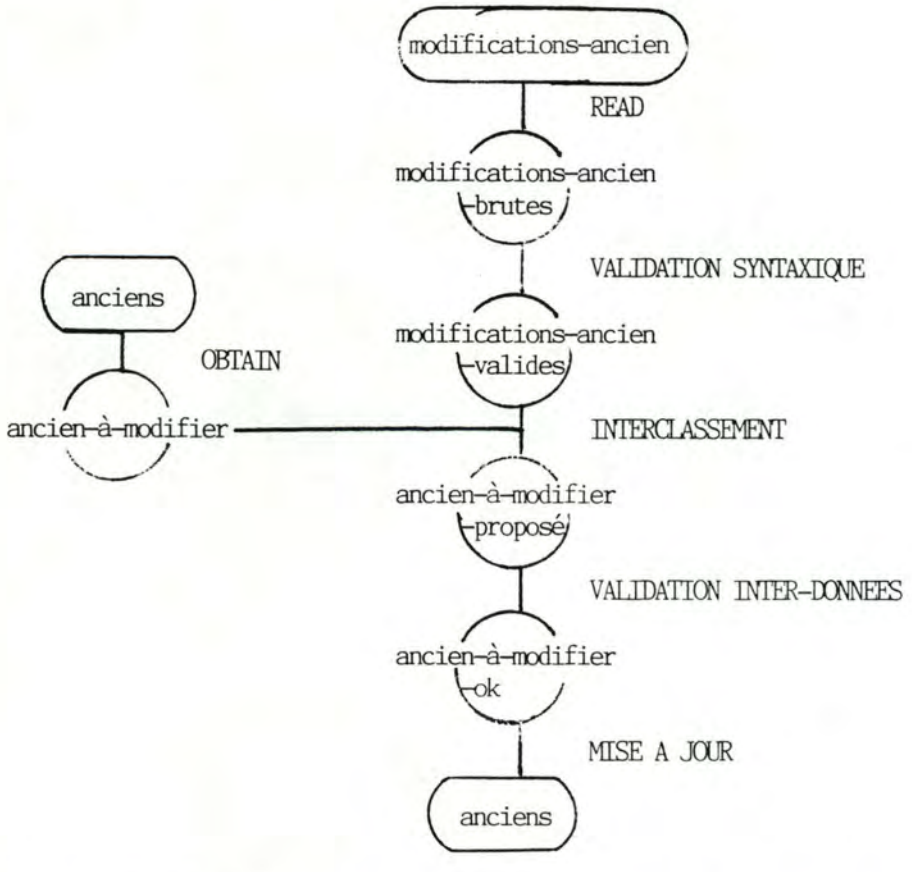


figure 5.10. Fonction MODIFIER ancien en batch.

En interactif, on procéderait de la manière suivante :
- demande et lecture de la clé de l'ancien à modifier
- extraction et affichage de cet ancien tel qu'il existe dans le signalétique
- demande avec validation syntaxique des données nouvelles au terminal (l'interclassement est donc inclus dans l'acquisition)
- validation globale de l'ancien proposé
- et mise à jour, si c'est correct.

Ceci est présenté à la figure 5.11.

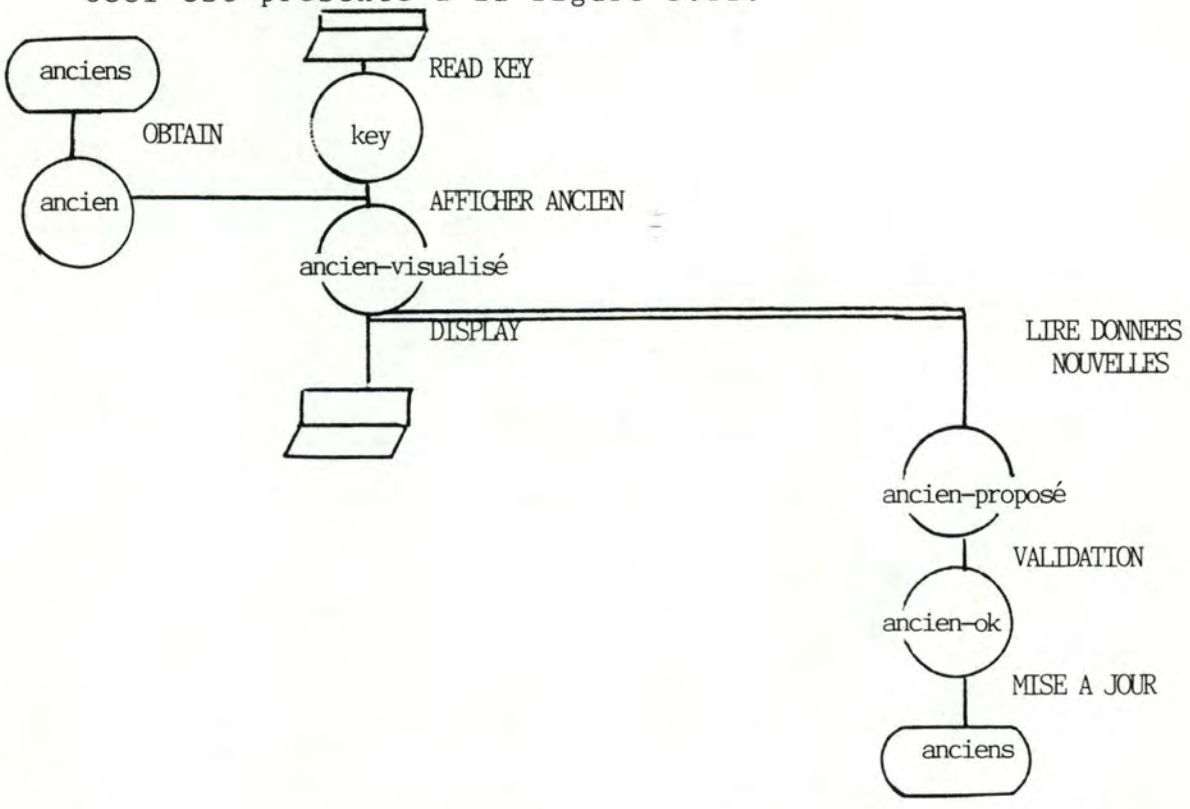


figure 5.11. Fonction MODIFIER ancien en interactif.

On aurait pu, en interactif, agir de la même manière qu'en batch, mais on n'aurait pas profité dans ce cas de l'interactivité avec les données déjà présentes dans le signalétique.

5.2. MODE TRANSACTIONNEL.

5.2.1. Définition du mode transactionnel.

Le mode transactionnel est un mode d'exploitation défini pour garantir l'intégrité de la base de données (BD) (1) dans un contexte d'accès concurrent à cette BD. Ce mode d'exploitation peut être surimposé au mode batch ou au mode interactif.

Dans l'accès concurrent à la BD, plusieurs processus peuvent accéder "simultanément" à une BD, notamment pour modifier un enregistrement ou un groupe d'enregistrements. Toute opération de modification entreprise par un processus P1 prend un certain laps de temps à s'exécuter; si entre-temps, un autre processus P2 accède à la BD, il a en réalité accès à une BD dont l'état est incertain. Il faut donc lui interdire l'accès tant que dure l'opération de mise à jour entamée par P1. C'est l'exclusion mutuelle. L'exclusion mutuelle ne peut être résolue qu'en dehors des programmes d'application. Ce problème ne relève donc pas de la méthode Memo-Proges mais du système d'exploitation. Sur Texas Instruments (2), il existe des primitives de contrôle de l'accès concurrent disponibles dans le langage de programmation Cobol qui permettent de verrouiller ou de libérer un enregistrement. Ces primitives résolvent l'accès concurrent à un enregistrement. Pour résoudre le problème de l'accès concurrent à un groupe d'enregistrements, le concept de transaction a été introduit.

Une transaction est une séquence d'opérations élémentaires de transformation de la BD, séquence telle que, soit toutes ces opérations aboutissent à leur terme normal, soit aucune ne produit d'effet. Par exemple, déclarer la procédure

---

(1) Par base de données, nous entendons l'ensemble des fichiers communément accessibles à un groupe de processus. On notera que le sens de ce terme n'est pas ici limité à un mode spécial de stockage des données.

(2) C'est sur cet ordinateur que nous avons implémenté le cas PETITPAS dans un contexte interactif multi-utilisateurs.

d'enregistrement d'un bon de commande comme étant une transaction, c'est poser la règle qu'un bon de commande ne sera jamais enregistré partiellement : ou bien il le sera complètement, ou bien rien du tout ne sera enregistré.
Dans le cas général d'une transaction portant sur plusieurs enregistrements de la base de données, trois primitives sont employées : "début/fin de transaction" et "annuler la transaction".
Soit une transaction TR modifiant la partie Dn de la BD. Pendant tout le déroulement de TR, la partie Dn tout entière doit être placée en accès exclusif. Dans ce but, toute opération de consultation de la BD entreprise dans le cadre de TR vérifie d'abord que l'enregistrement demandé n'est pas déjà verrouillé à l'usage d'un autre processus; si l'enregistrement est libre, elle le verrouille; sinon, elle provoque automatiquement l'annulation de TR. En outre, tout enregistrement inséré dans la BD est également verrouillé. A la sortie de TR ("fin TR" ou "annuler TR"), tous ces enregistrements sont automatiquement libérés.
En un point quelconque de son déroulement, TR peut constater une condition d'abandon; elle provoque alors, par l'appel explicite de "annuler TR", sa propre annulation. Une annulation sera également implicitement provoquée lors de tout incident survenant pendant les opérations de sortie exécutées par "fin TR".
Annuler une transaction, c'est restaurer la BD dans l'état où elle se trouvait au moment de l'entrée "début TR". A cette fin, toute opération de mise à jour d'un enregistrement est précédée de la création, dans un journal ("log"), d'une "image avant" de cet enregistrement (une opération d'insertion est précédée de la création d'une image "néant").
Par ailleurs, la primitive "début TR" produit également dans le journal un repère. L'annulation d'une transaction consiste simplement à réappliquer dans la BD les "images avant" constituées depuis le dernier repère "début TR";
Cette opération a reçu le nom de "roll back".

Si, en plus des "images avant", sont produites des "images après" modification, et si la primitive "fin TR" produit elle aussi un repère ("commit"), un fichier altéré pourra être restauré par application sur une copie précédente de la BD de toutes les "images après" situées entre un repère "début TR" et un repère "commit" de transaction aboutie normalement. Cette procédure est appelée procédure de recouvrement ("recovery").
Ces primitives n'existent pas dans Memo-Proges. L'impact de telles primitives dans Memo-Proges mériterait une étude approfondie.

5.2.2. Système d'exploitation et mode transactionnel.

Dans un environnement transactionnel, la distribution des messages dans le temps et l'espace est aléatoire, et les messages ont -globalement- des points d'origine et de destination multiples. Il peut exister simultanément plusieurs demandes d'exécution d'une même logique, c'est-à-dire du même texte programmé. Ceci doit être géré par le système d'exploitation.
Deux stratégies sont possibles : (1) ou bien l'on crée autant de processus programmés qu'il existe de demandes, en chargeant en mémoire centrale plusieurs copies du même texte programmé - la logique de celui-ci est "mono-file d'attente"; (2) ou bien le sous-système allocateur aiguille les différents messages vers le même processus programmé - le programme a alors une logique "multi-file d'attente".

Programme multi-file d'attente et moniteur de transaction.

Un programme multi-file d'attente doit généralement pouvoir distinguer les auteurs des messages de données qui lui parviennent; en outre, dans le cas où une succession de plusieurs messages d'entrée sont nécessaires pour pouvoir mener une transaction à son terme (par exemple, l'enregistrement d'un bon de commande ligne à ligne), il devra

par lui-même constituer et gérer des files d'attente internes pour le compte de chacun de ses utilisateurs.
Pour faciliter ces tâches particulièrement ardues, les systèmes d'exploitation utilisant cette stratégie offrent un moniteur de transactions paramétrable qui les prend en charge. Pour le système d'exploitation, le moniteur de transactions est un processus programmé utilisateur ordinaire (au même titre qu'un programme exécuté en mode "batch") et les programmes d'application ne sont que des sous-routines de ce processus. Les moniteurs de transactions résolvent en même temps le problème de l'accès concurrent aux données : le processus moniteur prend l'accès exclusif sur les fichiers et, par une programmation interne qui lui est propre, distribue aux "sous-routines" d'application l'accès aux enregistrements. Il n'était pas envisageable de travailler dans un tel contexte, car nous aurions été contraintes par des conventions de liaison des ous-routines, conventions propres à chaque ordinateur.

Programme multi-file d'attente et code réentrant.

Pour diminuer l'encombrement de la mémoire centrale lors de la duplication des processus, les compilateurs et éditeurs de liens actuels (entre autres sur Texas Instruments et Digital) créent du "code réentrant" (1). Qu'est-ce à dire ? Dans l'espace de mémoire centrale occupé par un programme, certaines parties ne se modifient pas (données constantes, texte des instructions); d'autres se modifient (zones de données variables). Un code réentrant est la scission (par le compilateur et l'éditeur de liens) du texte programmé en deux segments : un segment invariable et un segment variable. Plusieurs processus programmés simultanés peuvent utiliser en mémoire centrale une copie unique du segment invariable, qualifié pour cette raison de "réentrant"; chaque processus programmé possèdera cependant en propre une copie distincte du segment variable.

CONCLUSION.

Nous avons voulu établir, au cours de ce mémoire, une méthode d'analyse réalisant un continuum entre l'analyse conceptuelle et l'analyse organique. Cet objectif se situe dans le cadre d'une démarche méthodologique industrielle en vue d'une génération automatique d'une solution implémentable à partir d'une solution conceptuelle.

Pour réaliser ce continuum, il faut se définir des modèles pour l'analyse conceptuelle et pour l'analyse organique et transformer les premiers en les seconds. Nous avons, pour notre part, utilisé des modèles et outils existants :
- modèles de l'analyse conceptuelle de Bodart, modèles qui nous étaient familiers, et
- modèles de l'analyse organique de Clarinval, modèles que nous avons appris à connaître.

Nous avons opéré la réinterprétation des modèles IDA [Bodart-Pigneur] en modèles Memo-Proges [Clarinval].

Cette réinterprétation a été réalisée, du point de vue des traitements, pour les modèles statique et dynamique de Bodart. Cette réinterprétation a été aisée, car bien que les méthodes aient été développées indépendamment l'une de l'autre, leurs concepts ne sont pas éloignés. En effet, elles considèrent toutes les deux un traitement comme une transformation d'informations :
- Bodart voit principalement un traitement comme une transformation de messages, tandis que
- Clarinval voit un traitement comme une transformation de fichiers virtuels.

Quant à la réinterprétation du modèle Entité-Association, étant donné qu'il n'existait pas dans Memo-Proges de modèle de structuration des données, nous avons arbitrairement proposé des règles de réinterprétation. Ces règles ont été définies dans un cadre d'organisation de données en fichiers. A notre avis, cet aspect reste pendant.

La méthode d'analyse que nous avons proposée, s'est voulue générale pour autant qu'on puisse l'être en traitant dans le détail deux problèmes. Dans ce souci de généralité, nous avons appliqué les mêmes modèles et outils aux deux types de problèmes. Nous pensons que ces modèles sont très bien adaptés à l'analyse des problèmes avec flux d'information. Quant à l'analyse des problèmes sans flux, il se pose le dilemme suivant : "Il n'y a pas de flux d'information au niveau conceptuel; cependant il y aura un flux au niveau organique".

Enfin, nous avons constaté que les modes d'exploitation n'étaient pas neutres par rapport aux solutions obtenues. Nous avons mis en évidence l'influence de l'utilisation d'un mode d'exploitation sur la conception même des programmes. Nous sommes cependant convaincues que l'impact du choix batch/interactif ne se limite pas à ce domaine. Ce choix a des répercussions, qu'il ne faudrait pas négliger, sur l'organisation dans laquelle le système d'information est mis en place et sur les personnes qui la composent.

BIBLIOGRAPHIE.

[Bodart-Lesuisse] F. BODART, R. LESUISSE : Une méthode d'analyse oérationnelle en informatique d'organisation,
in Proc. Colloque AFCET, Paris, 1982.

[Bodart-Pigneur] F. BODART, Y. PIGNEUR : Conception assistée des applications informatiques :
1.- Etude d'opportunité et analyse conceptuelle,
Masson, Presses Universitaires de Namur, 1983.

[Chen] P.P. CHEN : The Entity-Relationship Model. Toward a Unified View of Data,
in ACM TODS, Vol. 1, nr 1, 1976.

[Clarinval.a] A. CLARINVAL : Memo-Proges. Méthode Modulaire de Programmation pour l'Informatique de Gestion,
Thèse de Doctorat en Sciences (option informatique) aux Facultés Universitaires Notre-Dame de la Paix - Namur, octobre 1977.

[Clarinval.b] A. CLARINVAL : Introduction au concept d'informatique transactionnelle,
Document interne, Groupe S, octobre 1983.

[Liskov] Barbara H. LISKOV, Stephen N. ZILLES : Spécification TEchniques for Data Abstractions,
in IEEE Transactions on Software Engineering, Vol. SE-1, nr 1, mars 1975.

[Parnas] David L. PARNAS : Software Engineering Principles,
Document reçu lors du Colloque "Software Engineering", Leuven, mai 1984.

[Petitpas] Listing des spécifications D.S.L. du cas PETITPAS,
Document interne à l'Institut d'Informatique des Facultés Universitaires Notre-Dame de la Paix -Namur, 1981.

- ANNEXE 1 -

Définition du langage D.S.L.

Properties common to each object-type

```
SYNONYMS ARE synonym-name (, synonym-name );

KEYWORDS ARE string (, string );

ATTRIBUTES ARE Attr-name  {sys-par}  (, Attr-name  {sys-par} );
                          {integer}                {integer}
                          {real   }                {real   }
                          {string }                {string }

DESCRIPTION;
    comment entry;

SEE-MEMO Memo-name (, Memo-name );
```

INTERFACE SECTION

```
DEFINE INTERFACE Interface-name;

_____________________________________STATICS_____________________________________

PRODUCES [sys-par] COPIES OF Message-name (, [sys-par] COPIES OF Message-name );
         [integer]                          [integer]

GENERATES [[sys-par]] Message-name (, [[sys-par]] Message-name );
          [real    ]                  [real    ]
          [integer]                   [integer]

GETS [sys-par] COPIES OF Message-name (, [sys-par] COPIES OF Message-name );
     [integer]                           [integer]

RECEIVES [[sys-par]] Message-name (, [[sys-par]] Message-name );
         [integer]                   [integer]
         [real    ]                  [real    ]

_____________________________________SYSTEM-STRUCTURE_____________________________

PART OF Interface-name;

SUBPARTS ARE Interface-name (, Interface-name );

_____________________________________DYNAMICS_____________________________________

CAUSES HAPPENING OF Message-name [  EVERY   [sys-par      ] ] DURING Calendar-name;
                                 [          [time-constant] ]
                                 [ [sys-par] TIMES          ]
                                 [ [real   ]                ]
                                 [ [integer]                ]

MANAGES [Process-name             ]  (, [Process-name             ] );
        [Resource-name            ]     [Resource-name            ]
        [Consumable-Resource-name ]     [Consumable-Resource-name ]
```

MESSAGE Section

```
DEFINE MESSAGE Message-name;

LAYOUT;
   Comment entry;
_____________________________________STATICS_____________________________________
PRODUCED IN {sys-par} COPIES BY {Process-name   };
           {integer}          {Interface-name}

GENERATED [{sys-par} TIMES] BY Process-name [IF [NOT] Condition-name];
          {integer}
          {real   }

GENERATED [{sys-par} TIMES] BY Interface-name;
          {integer}
          {real   }

GET IN {sys-par} COPIES BY {Process-name   }
      {integer}          {Interface-name}

RECEIVED [{sys-par} TIMES] BY {Process-name   } (, [{sys-par} TIMES] BY {Process-name   } );
         {integer}          {Interface-name}     {integer}          {Interface-name}
         {real   }                               {real   }

_____________________________________SYSTEM-STRUCTURE_____________________________

PART OF Message-name;

SUBPARTS ARE Message-name (, Message-name );

_____________________________________DATA-STRUCTURE_______________________________

SAME DATA STRUCTURE AS Message-name;

CONSISTS OF [{sys-par}] {Element-name } (, [{sys-par}] {Element-name } );
            {integer}  {Group-name   }     {integer}  {Group-name   }
            {real   }  {Entity-name  }     {real   }  {Entity-name  }
                       {Relation-name}                 {Relation-name}

_____________________________________DYNAMICS_____________________________________

HAPPENS {  EVERY  {sys-par      } } FROM Interface-name DURING Calendar-name;
        {         {time-constant} }
        { {sys-par} TIMES         }
        { {integer}               }
        { {real   }               }

ON GENERATION CONTRIBUTES TO [MANY] Synchro-pt-name (, [MANY] Synchro-pt-name ) [IF [NOT] Condition-nar


ON GENERATION TRIGGERS Process-name [FOR EACH Attr-name] [IF [NOT] Condition-name];
```

SYNCHRONIZATION-POINT Section

```
DEFINE SYNCHRONIZATION-POINT Synchro-pt-name;

REALIZED-WHEN;
      comment entry;

_____________________________________DYNAMICS_____________________________________

TRIGGERS Process-name [FOR EACH Attr-name] [IF [NOT] Condition-name];

CONTRIBUTES TO [MANY] Synchro-pt-name (, [MANY] Synchro-pt-name ) [IF [NOT] Condition-name];

CONTRIBUTED [MANY TIMES] BY {GENERATION   OF Message-name (, Message-name )      }
                            {INCEPTION    OF Process-name (, Process-name )      }
                            {TERMINATION  OF Process-name (, Process-name )      }
                            {REALIZATION  OF Synchro-pt-name (, Synchro-pt-name )} [IF [NOT] Condition-n

MEMORIZES {GENERATION OF Message-name     } [{UNTIL  {INCEPTION OF Process-name     } }];
          {INCEPTION OF Process-name      }  {       {TERMINATION OF Process-name   } }
          {TERMINATION OF Process-name    }  {       {REALIZATION OF Synchro-pt-name} }
          {REALIZATION OF Synchro-pt-name }  {       {GENERATION OF Message-name    } }
                                             { DURING  time-constant                  }

DO NOT MEMORIZE {GENERATION OF Message-name     };
                {INCEPTION OF Process-name      }
                {TERMINATION OF Process-name    }
                {REALIZATION OF Synchro-pt-name }
```

CONDITION Section

```
DEFINE CONDITION Condition-name;

TRUE-WHILE;
      Comment entry;

PROBABILITY TRUE IS {sys-par};
                    {integer}
                    {real   }
```

PROCESS Section

```
DEFINE PROCESS Process-name;

_____________________________________STATICS_____________________________________

GENERATES [{sys-par}] Message-name (, [{sys-par}] Message-name ) [IF [NOT] Condition-name];
          {integer}                   {integer}
          {real   }                   {real   }

RECEIVES [{sys-par}] Message-name (, [{sys-par}] Message-name );
         {integer}                   {integer}
         {real   }                   {real   }

PRODUCES {sys-par} COPIES OF Message-name (, {sys-par} COPIES OF Message-name );
         {integer}                           {integer}

GETS {sys-par} COPIES OF Message-name (, {sys-par} COPIES OF Message-name );
     {integer}                           {integer}

USES {Set-name      } (, {Set-name      } ) [TO DERIVE {Set-name      } (, {Set-name      } )];
     {Entity-name   }    {Entity-name   }              {Entity-name   }    {Entity-name   }
     {Relation-name}    {Relation-name}               {Relation-name}    {Relation-name}
     {Group-name    }    {Group-name    }              {Group-name    }    {Group-name    }
     {Element-name  }    {Element-name  }              {Element-name  }    {Element-name  }

DERIVES {Set-name      } (, {Set-name      } ) [USING {Set-name      } (, {Set-name      } )];
        {Entity-name   }    {Entity-name   }          {Entity-name   }    {Entity-name   }
        {Relation-name}    {Relation-name}           {Relation-name}    {Relation-name}
        {Group-name    }    {Group-name    }          {Group-name    }    {Group-name    }
        {Element-name  }    {Element-name  }          {Element-name  }    {Element-name  }

ADDS [{sys-par}] {Entity-name   } (, [{sys-par}] {Entity-name   } );
     {integer}  {Relation-name}      {integer}  {Relation-name} )
     {real   }                       {real   }

REMOVES [{sys-par}] {Entity-name   } (, [{sys-par}] {Entity-name   } );
        {integer}  {Relation-name}      {integer}  {Relation-name} )
        {real   }                       {real   }

REFERENCES [{Element-name} (, {Element-name} ) INTO]
           {Group-name   }    {Group-name   }

          [{sys-par}] {Entity-name   } (, [{sys-par}] {Entity-name   } );
           {integer}  {Relation-name}      {integer}  {Relation-name}
           {real   }                       {real   }

MODIFIES [{Element-name} (, {Element-name} ) INTO]
         {Group-name   }    {Group-name   }

         [{sys-par}] {Entity-name   } (, [{sys-par}] {Entity-name   } );
          {integer}  {Relation-name}      {integer}  {Relation-name}
          {real   }                       {real   }


PROCEDURE;
   comment entry;
```

SYSTEM-STRUCTURE

```
PART OF Process-name;

SUBPARTS ARE Process-name (, Process-name );
```

DYNAMICS

```
ON {INCEPTION   } TRIGGERS Process-name [FOR EACH Attr-name] [IF [NOT] Condition-name];
   {TERMINATION}

TRIGGERED BY {INCEPTION   } OF Process-name [FOR EACH Attr-name] [IF [NOT] Condition-name];
             {TERMINATION} OF

TRIGGERED BY GENERATION OF Message-name [FOR EACH Attr-name] [IF [NOT] Condition-name];

TRIGGERED BY Synchro-pt-name [FOR EACH Attr-name] [IF [NOT] Condition-name];

ON {INCEPTION   } CONTRIBUTES TO [MANY] Synchro-pt-name (, [MANY] Synchro-pt-name ) [IF [NOT] Condition
   {TERMINATION}


UTILIZES Process-name (, Process-name );
UTILIZED BY Process-name (, Process-name );

[PREEMPTIVE] PRIORITY is {sys-par};
                         {integer}
                         {real   }

PERFORMED DURING {time-constant};
                 {sys-par      }

REQUIRES [{sys-par}] Resource-name (, [{sys-par}] Resource-name );
         {integer}                    {integer}
         {real   }                    {real   }

SHARES Resource-name (, Resource-name );

CONSUMES [{sys-par}] Consumable-Resource-name
         {integer}
         {real   }
      (, [{sys-par}] Consumable-Resource-name ) [PER time-constant];
         {integer}
         {real   }

MANAGED BY Interface-name (, Interface-name );
```

PROCESSOR/RESOURCE Section

```
DEFINE {RESOURCE} resource-name;
       {PROCESSOR }

_____________________________________SYSTEM-STRUCTURE_____________________________

PART OF Resource-name;

SUBPARTS ARE Resource-name (, Resource-name );

_____________________________________SYSTEM-ARCHITECTURE___________________________

UNIT PRICE IS {sys-par} Unit-name PER time-constant;
              {integer}
              {real   }

MEASURED BY Unit-name;

CAPACITY IS {sys-par};
            {integer}
            {real   }

SHARABLE AMONG {sys-par};
               {integer}
               {real   }

AVAILABLE DURING Calendar-name [FOR Process-name] (, Calendar-name [FOR Process-name] );

REQUIRES [{sys-par}] Resource-name (, [{sys-par}] Resource-name );
         {integer}                    {integer}
         {real   }                    {real   }

SHARES Resource-name (, Resource-name );

CONSUMES [{sys-par}] Consumable-Resource-name
         {integer}
         {real   }
      (, [{sys-par}] Consumable-Resource-name ) [PER time-constant];
         {integer}
         {real   }

REQUIRED BY {Process-name } (, {Process-name } ) [AT RATE OF {sys-par}]
            {Resource-name}    {Resource-name}               {integer}
                                                             {real   }

SHARED BY {Process-name } (, {Process-name } );
          {Resource-name}    {Resource-name}

MANAGED BY Interface-name (, Interface-name );
```

CONSUMABLE-RESOURCE Section

```
DEFINE CONSUMABLE-RESOURCE Consumable-Resource-name;

_____________________________________SYSTEM-STRUCTURE_____________________________

PART OF Consumable-Resource-name;

SUBPARTS ARE Consumable-Resource-name (, Consumable-Resource-name );

_____________________________________SYSTEM-ARCHITECTURE___________________________

UNIT PRICE IS {sys-par} Unit-name;
              {integer}
              {real   }

MEASURED BY Unit-name;

AVAILABILITY IS {sys-par} PER Time-constant;
                {integer}
                {real   }

AVAILABLE DURING Calendar-name [FOR Process-name] (, Calendar-name [FOR Process-name] );

CONSUMED BY {Process-name } (, {Process-name } )
           {Resource-name}     {Resource-name}
        [AT RATE OF {sys-par}] [PER time-constant];
                    {integer}
                    {real   }


MANAGED BY Interface-name (, Interface-name );
```

ENTITY Section

DEFINE ENTITY Entity-name

STATICS

```
USED BY Process-name (, Process-name ) [TO DERIVE {Set-name      } (, {Set-name      } )];
                                                  {Entity-name   }     {Entity-name   }
                                                  {Relation-name }     {Relation-name }
                                                  {Group-name    }     {Group-name    }
                                                  {Element-name  }     {Element-name  }

DERIVED BY Process name (, Process-name ) [USING {Set-name      } (, {Set-name      } )];
                                                {Entity-name   }     {Entity-name   }
                                                {Relation-name }     {Relation-name }
                                                {Group-name    }     {Group-name    }
                                                {Element-name  }     {Element-name  }

ADDED BY Process-name (, Process-name );

REMOVED BY Process-name (, Process-name );

[HAS {Element-name} (, {Element-name} )] MODIFIED BY Process-name (, Process-name );
     {Group-name   }     {Group-name   }

[HAS {Element-name} (, {Element-name} )] REFERENCED BY Process-name (, Process-name );
     {Group-name   }     {Group-name   }
```

DATA-STRUCTURE

```
SUBTYPES ARE Entity-name (, Entity-name );

SUBTYPE OF Entity-name (, Entity-name );

ROLE NAME IS string IN Relation-name (, string IN Relation-name );

COLLECTED IN Set-name (, Set-name );

CONTAINED [{sys-par} TIMES] IN Message-name (, [{sys-par} TIMES] IN Message-name ,,
          {real    }                           {real    }
          {integer }                           {integer }

RELATED VIA Relation-name;

CONSISTS OF [{sys-par}] {Element-name} (, [{sys-par}] {Element-name} );
            {real    } {Group-name   }     {real    } {Group-name   }
            {integer }                     {integer }

IDENTIFIED BY {Element-name}, (, {Element-name} ) [VIA Relation-name];
              {Group-name  }      {Group-name   }

CARDINALITY IS {sys-par} IN Set-name (, {sys-par} IN Set-name ) [DURING Calendar-name];
               {integer }               {integer }
               {real    }               {real    }
```

RELATION Section

```
DEFINE RELATION Relation-name;

_____________________________________STATICS_____________________________________

USED BY Process-name (, Process-name ) [TO DERIVE {Set-name      } (, {Set-name      }] );
                                                  {Entity-name   }     {Entity-name   }
                                                  {Relation-name }     {Relation-name }
                                                  {Group-name    }     {Group-name    }
                                                  {Element-name  }     {Element-name  }

DERIVED BY Process-name (, Process-name ) [USING {Set-name      } (, {Set-name      }] );
                                                 {Entity-name   }     {Entity-name   }
                                                 {Relation-name }     {Relation-name }
                                                 {Group-name    }     {Group-name    }
                                                 {Element-name  }     {Element-name  }

ADDED BY Process-name (, Process-name );

REMOVED BY Process-name (, Process-name );

[HAS {Element-name} (, {Element-name} )] MODIFIED BY Process-name (, Process-name );
     {Group-name   }     {Group-name   }

[HAS {Element-name} (, {Element-name} )] REFERENCED BY Process-name (, Process-name );
     {Group-name   }     {Group-name   }

_____________________________________DATA-STRUCTURE_____________________________

ROLE-NAME is string FOR Entity-name (, string FOR Entity-name );

COLLECTED IN Set-name (, Set-name );

CONTAINED [{sys-par} TIMES] IN Message-name (, [{sys-par} TIMES] IN Message-name );
          {integer}                            {integer}
          {real   }                            {real   }

RELATES Entity-name (, Entity-name );

CONSISTS OF [{sys-par}] {Element-name} (, [{sys-par}] {Element-name} );
            {integer}  {Group-name   }     {integer}  {Group-name   }
            {real   }                      {real   }

IDENTIFIED BY {Element-name} (, {Element-name} );
              {Group-name   }     {Group-name   }

CARDINALITY IS {sys-par} IN Set-name (, {sys-par} IN Set-name ) [DURING Calendar-name];
               {integer}                {integer}
               {real   }                {real   }

CONNECTIVITY IS {sys-par} FOR Entity-name (, {sys-par} FOR Entity-name );
                {integer}                    {integer}
                {string }                    {string }
```

ELEMENT Section

```
DEFINE ELEMENT Element-name;

_____________________________________STATICS_____________________________________

USED BY Process-name (, Process-name ) [TO DERIVE {Set-name     } (, {Set-name     }] );
                                                  {Entity-name  }    {Entity-name  }
                                                  {Relation-name}    {Relation-name}
                                                  {Group-name   }    {Group-name   }
                                                  {Element-name }    {Element-name }

DERIVED BY Process-name (, Process-name ) [USING {Set-name     } (, {Set-name     }] );
                                                 {Entity-name  }    {Entity-name  }
                                                 {Relation-name}    {Relation-name}
                                                 {Group-name   }    {Group-name   }
                                                 {Element-name }    {Element-name }

REFERENCED IN {Entity-name  } BY Process-name (, Process-name );
              {Relation-name}

MODIFIED IN {Entity-name  } BY Process-name (, Process-name );
            {Relation-name}

_____________________________________DATA-STRUCTURE_____________________________

CODE {integer} MEANS {integer} [THRU {integer}] (, {integer} [THRU {integer}] );
     {real   }       {real   }       {real   }     {real   }       {real   }
     {string }       {string }       {string }     {string }       {string }

DOMAIN OF VALUES ARE {sys-par} [THRU {sys-par}] (, {sys-par} [THRU {sys-par}] );
                     {integer}       {integer}     {integer}       {integer}
                     {real   }       {real   }     {real   }       {real   }
                     {string }       {string }     {string }       {string }


SAME DOMAIN AS Element-name (, Element-name )

CONTAINED [{sys-par} TIMES] IN {Entity-name  } (, [{sys-par} TIMES] IN {Entity-name  } );
          {integer}            {Relation-name}     {integer}            {Relation-name}
          {real   }            {Message-name }     {real   }            {Message-name }
                               {Group-name   }                          {Group-name   }

FORMAT IS string;


FUNCTIONALLY DETERMINES {Element-name} FOR {Entity-name  } (, {Element-name} FOR {Entity-name  } );
                        {Group-name  }     {Relation-name}    {Group-name  }     {Relation-name}

FUNCTIONALLY DEPENDENT OF {Element-name} FOR {Entity-name  } (, {Element-name} FOR {Entity-name  }
                          {Group-name  }     {Relation-name}    {Group-name  }     {Relation-name}

IDENTIFIES Entity-name (, Entity-name ) [VIA Relation-name];

IDENTIFIES Relation-name (, Relation-name );
```

```
DEFINE SET Set-name;

_____________________________________STATICS_____________________________________

USED BY Process-name (, Process-name ) [TO DERIVE {Set-name      } (, {Set-name      }] );
                                                  {Entity-name   }     {Entity-name   }
                                                  {Relation-name}     {Relation-name}
                                                  {Group-name    }     {Group-name    }
                                                  {Element-name  }     {Element-name  }

DERIVED BY Process-name (, Process-name ) [USING {Set-name      } (, {Set-name      }] );
                                                 {Entity-name   }     {Entity-name   }
                                                 {Relation-name}     {Relation-name}
                                                 {Group-name    }     {Group-name    }
                                                 {Element-name  }     {Element-name  }

_____________________________________DATA-STRUCTURE_____________________________

COLLECTION OF {Relation-name} (, {Relation-name} );
              {Entity-name   }    {Entity-name   }

SUBSETS ARE Set-name (, Set-name );

SUBSETS OF Set-name (, Set-name );

CARDINALITY IS {sys-par} FOR {Relation-name} (, {sys-par} FOR {Relation-name} ) [DURING Calendar-na
               {integer}     {Entity-name   }    {integer}     {Entity-name   }
               {real   }                         {real   }
```

ATTRIBUTE Section

```
DEFINE ATTRIBUTE Attr-name;

VALUES ARE {sys-par} FOR non- {Attr-name} (, {sys-par} FOR non- {Attr-name} );
           {integer}          {sys-par  }    {integer}          {sys-par  }
           {real   }                         {real   }
           {string }                         {string }

PROTOTYPING-VALUE;
          Comment-entry;

SIMULATION VALUE IS (, {sys-par} ON {INCEPTION OF Process-name  } );
                       {integer}    {TERMINATION OF Process-name}
                       {real   }    {GENERATION OF Message-name }
                       {string }    {REALIZATION OF Synchro-name}
```

MEMO Section

```
DEFINE MEMO Memo-name;

APPLIES TO non-Memo-name (, non-Memo-name );
```

UNIT Section

```
DEFINE UNIT Unit-name;

EQUIVALENT TO {sys-par} Unit-name (, {sys-par} Unit-name );
              {integer}              {integer}
              {real   }              {real   }
```

GROUP Section

```
DEFINE GROUP Group-name;

_____________________________________STATICS_____________________________________

USED BY Process-name (, Process-name ) [TO DERIVE {Set-name     } (, {Set-name     }] );
                                                  {Entity-name  }    {Entity-name  }
                                                  {Relation-name}    {Relation-name}
                                                  {Group-name   }    {Group-name   }
                                                  {Element-name }    {Element-name }

DERIVED BY Process-name (, Process-name ) [USING {Set-name     } (, {Set-name     }] );
                                                 {Entity-name  }    {Entity-name  }
                                                 {Relation-name}    {Relation-name}
                                                 {Group-name   }    {Group-name   }
                                                 {Element-name }    {Element-name }

REFERENCED IN {Entity-name  } BY Process-name (, Process-name );
              {Relation-name}

MODIFIED IN {Entity-name  } BY Process-name (, Process-name );
            {Relation-name}

_____________________________________DATA-STRUCTURE_____________________________

CONTAINED [{sys-par} TIMES] IN {Entity-name  } (, [{sys-par} TIMES] IN {Entity-name  } );
          {integer}            {Relation-name}     {integer}           {Relation-name}
          {real   }            {Message-name }     {real   }           {Message-name }
                               {Group-name   }                         {Group-name   }

CONSISTS OF [{sys-par}] {Element-name}
            {integer}   {Group-name  }
            {real   }

       (, [{sys-par}] {Element-name} ) [VIA Relation-name];
          {integer}   {Group-name  }
          {real   }

FUNCTIONALLY DETERMINES {Element-name} FOR {Entity-name  } (, {Element-name} FOR {Entity-name  } );
                        {Group-name  }     {Relation-name}    {Group-name  }     {Relation-name}

FUNCTIONALLY DEPENDENT OF {Element-name} FOR {Entity-name  } (, {Element-name} FOR {Entity-name
                          {Group-name  }     {Relation-name}    {Group-name  }     {Relation-name}

IDENTIFIES Entity-name (, Entity-name ) [VIA Relation-name];

IDENTIFIES Relation-name (, Relation-name );
```

SYSTEM-PARAMETER Section

```
DEFINE SYSTEM-PARAMETER sys-par;

IS A FUNCTION OF sys-par DURING Calendar-name (, sys-par DURING Calendar-name );

DETERMINES sys-par DURING Calendar-name (, sys-par DURING Calendar-name );

VALUE IS {time-constant};
         {integer      }
         {real         }
         {string       }

RANGE IS {Time-constant} [THRU {time-constant}]
         {integer      }       {integer      }
         {real         }       {real         }
         {string       }       {string       }
      (, {Time-constant} [THRU {time-constant}] ) [WITH PROBABILITY {sys-par}]
         {integer      }       {integer      }                      {integer}
         {real         }       {real         }                      {real   }
         {string       }       {string       }

DISTRIBUTION IS {ERLANG } WITH PARAMETER {integer      } [AND {integer      }],
                {NEGEXP }                {real         }      {real         }
                {NORMAL }                {time-constant}      {time-constant}
                {POISSON}
                {UNIFORM}
```

CALENDAR Section

```
DEFINE CALENDAR Calendar-name;

ACTIVE Hrs-interval-name (, Hrs-interval-name ) [ON Day-interval-name (, Day-interval-name )
```

DAY-INTERVAL Section

```
DEFINE DAY-INTERVAL Day-interval-name;

SPANS FROM integer TO integer DAY [PER {WEEK }];
                                  {MONTH}
```

HRS-INTERVAL Section

```
DEFINE HRS-INTERVAL Hrs-interval-name ;

SPANS FROM time-constant TO time-constant;
```

- ANNEXE 2 -

Présentation du cas PETITPAS.

CAS D'ANALYSE: FIRME "PETITPAS"

2- PROPOSITION D'AUTOMATISATION :

Le texte qui suit est une proposition de solution automatisee du traitement de Commandes Client et des livraisons Fournisseur.Pour ce dernier traitement, on se limitera a mentionner l'impact sur le niveau des stocks.

1.0 MODIFICATION DE LA POLITIQUE DE LIVRAISON.

Dans le cadre de la politique de limitation des ruptures de stocks et de l'amelioration du service a la clientele, la firme a decide de ne plus limiter le nombre de differes par commande.

En effet, l'analyse de la nouvelle solution a permis de mettre en evidence que, compte tenu de l'amelioration attendue de la gestion des stocks et des clauses de contrats de transport avec la SNCB et la Regie des Postes, il serait commercialement avantageux pour la firme d'admettre a priori pour une commande un nombre illimite de differes. Les livraisons s'effectueront au fur et a mesure des reapprovisionnements; la commande ne sera archivee qu'apres son apurement complet ou epuisement complet des articles restant a livrer.

Le paiement par cheque entrainant des couts administratifs eleves, la firme a decide de le supprimer et de generaliser le systeme d' expedition contre remboursement

2.0 NOUVELLE ORGANISATION DE LA FIRME.

La gestion en temps reel des stocks (enregistrement des commandes client et des livraisons fournisseurs) a provoque la reorganisation suivante de la firme:

1. La Direction Generale et les departements Commercial, Financier, et du Personnel restent inchanges.

2. Le departement de Production sera organise commu suit:

   2.1 Service d'enregistrement de commandes procedera a l'aide de terminaux

   - au controle des commandes
   - a la mise a jour du "fichier" des Adresses-client
   - a l'enregistrement de commandes

   2.2 Le service de preparation des commandes procedera a la recherche et a l'enlevement des produits par X series de Y commandes.

   2.3 Le service emballage et expedition procedera a l'aide de terminaux

   - a la reconstitution de la commande
   - au lancement de la facturation
   - a l'emballage et a l'expedition du colis

3. Departement Achats et Fournisseurs.

3.1 Service des Achats

Le service des achats se charge d'effectuer la gestion des fournisseurs: choix des fournisseurs, commandes aux fournisseurs, suivi des commandes et des livraisons, analyse de l'Etat des stocks et des commandes client. Ce service est egalement dote de terminaux.

3.2 Service de Gestion des stocks

Tous les vendredis midi, les services de preparation des commandes, d'emballage et d'expedition cessent leur travail habituel et procedent a un inventaire permanent. Ils rangent les rayons, comptabilisent le nombre d'articles de chaque rayon et signalent l'etat des stocks a un gestionnaire des stocks qui corrige eventuellement la quantite en stock dans le "fichier d'Etat des stocks".

4. Service d'entreposage.

Ce service s'occupe de la reception des livraisons fournisseurs, de leur controle et de leur enregistrement par terminal.
Il est aussi responsable du rangement dans les magasins des marchandises recues.

5. Un systeme informatique, base sur un mini-ordinateur ou un reseau de micro-ordinateurs executera:

1- en liaison temps reel avec le service d'enregistrement des commandes client, le controles et l'enregistrement de ces commandes

2- en liaison temps reel avec le service d'entreposage, le controle et l'enregistrement des livraisons fournisseur

3- en liaison temps reel avec le service d'emballage et d'expedition, l'edition de tous les documents d'expedition en ce compris la facture

4- en liaison temps reel avec le service des achats, une aide a la preparation des commandes fournisseur

Compte tenu des liaisons avec les autres services, il assumera egalement les traitements suivants:

1- mise a jour de l'etat des stocks et des ressources

2- gestion des commandes differees

3- optimisation des recherches en magasin (ceci afin d'optimiser les parcours).

3.0 LES NOUVELLES PROCEDURES

3.1 ENREGISTREMENT ET CONTROLE DES COMMANDES CLIENTS.

3.1.1 PREPARATION DU BON DE COMMANDE -

A chaque arrivee de courrier, ce poste recoit les bons de commande (inchanges par rapport a la solution manuelle). Il decachete les enveloppes et s'assure que les bons sont signes.
Les bons de commande non signes, sont transmis a la cellule de traitement des cas litigieux, les autres sont diriges vers les operatrices d'enregistrement des commandes.

3.1.2 ENREGISTREMENT ET CONTROLE DES BONS DE COMMANDE CLIENTS.

Ce poste se compose de quelques operatrices qui sont en liaison directe avec l'ordinateur grace a un terminal a ecran. Le principe est de composer a l'ecran le bon de commande controle et eventuellement corrige.
C'est seulement si cette operation a reussi que le bon de commande est enregistre et que s'effectueront les mises a jour du "fichier" Adresse-client.

L'operatrice effectue dans l'ordre les operations suivantes:

A- IDENTIFICATION DU CLIENT.

On distingue deux cas, selon que le bon de commande possede un numero de client non prospect preimprime ou selon que ce numero n'existe pas.

Dans le premier cas, l'operatrice procedera a une recherche dans le "fichier" Adresse-client pour verifier l'existence de ce numero et la concordance des adresses si ce numero existe.
Si le numero n'existe pas, elle effectuera une recherche sur les nom, prenom et adresse afin de verifier une eventuelle concordance.

Dans le second cas, l'operatrice procedera a une recherche sur nom prenom et adresse.

B- CONSTITUTION DU CORPS DE LA COMMANDE

L'operatrice constitue la commande ligne par ligne. Elle a pour objectif d'interpreter au mieux le bon de commande du client afin de perdre le moins de commande possible. Pourtant si elle ne peut interpreter une ligne, elle abandonne entierement la commande. La regle d'interpretation est d'accorder la priorite au libelle du produit par rapport au numero de reference.
On distingue deux cas, selon que la ligne possede ou non un numero de produit. L'operatrice procede a l'identification d'une ligne par une analyse de concordance des libelles.
Pour un produit identifie, en l'absence d'une quantite commandee mentionnee par le client, l'ordinateur affichera la quantite par defaut.

Si la commande ne peut etre acceptee, l'operatrice marque la ou les ligne(s) incorrectes sur le bon de commande; celle-ci sera reexpediee au client.

Si toutes les lignes du bon de commande ont pu etre interpretees, l'ordinateur affiche alors le prix total a payer par le client.
L'ecart entre le total calcule et celui du bon de commande est apprecie par l'operatrice, qui decide d'enregistrer ou de refuser la commande; en cas de refus, la commande est reexpediee au client.

3.2 PREPARATION ET ENREGISTREMENT DES COMMANDES AUX FOURNISSEURS.

Quotidiennement, le service des achats aux fournisseurs consulte le "fichier" d'Etat des stocks. Cette interrogation fournit une liste des stocks a reapprovisionner sur base des calculs suivants:

Niveau des ressources = Quantite en stock +
Quantite en commande chez les fournisseurs -
Quantite due pour les commandes differees.

Si le niveau des ressources est < ou = au Point de Commande, il y a decision de reapprovisionnement. La quantite a commander doit etre le plus petit multiple de la Quantite economique de commande superieur a la difference entre le point de commande et le niveau des ressources. Connaissant ces informations,

le service des achats consulte eventuellement le "fichier" des Fournisseurs et celui des Commandes Fournisseurs en cours, contacte les fournisseurs et etablit un bon de commande au fournisseur selectionne.

Par fournisseur, le gestionnaire des achats possede notamment les renseignements suivants relatifs aux produits:

- prix unitaire
- delai de livraison
- capacite de livraison
- pourcentage de remise
- couts de transport.

Chaque commande adressee a un fournisseur, possede un numero de commande attribue par compostage.

Ce bon de commande est alors enregistre dans le "fichier" des Commandes fournisseur en cours, et pour chaque produit commande la quantite commandee est ajoutee a la Quantite en commande du "fichier" Etat des stocks.

3.3 RECEPTION DE LA COMMANDE FOURNISSEUR.

Le service d'entreposage receptionne les livraisons en provenance des differents fournisseurs. Il effectue le controle de la marchandise:

A- En consultant en temps reel le "fichier" des Commandes fournisseur en cours, il verifie que la livraison correspond en tout ou en partie a une commande chez ce fournisseur.

B- En s'assurant que la marchandise est conforme aux normes de qualite exigees.

Au fur et a mesure des manipulations de la marchandise, celle-ci est acheminee vers les rayonnages dans le magasin. Ensuite, il met a jour le "fichier" des Commandes fournisseur en cours (en apurant les commandes les plus anciennes).

3.4 MISE A JOUR DE L'ETAT DES STOCKS.

3.4.1 TRAITEMENT DES ENTREES EN STOCK ‹MAJ+›.

Ce traitement est active chaque fois qu'une transaction "Produit reapprovisionne" est creee par le traitement de reception des commandes fournisseur.

La quantite emmagasinee (Quantite livree - Quantite refusee) est ajoutee a la quantite disponible en stock et soustraite de la quantite en commande aupres du fournisseur.

Apres la mise a jour du stock, l'ordinateur procede a la selection des commandes differees dont une ligne au moins est relative a un produit reapprovisionne.

3.4.2 TRAITEMENT DES SORTIES DE STOCK ‹MAJ-› ET ORDONNANCEMENT.

Ce traitement est active par deux types de transactions:

soit une commande enregistree par le service d'enregistrement des commandes
soit une commande differee selectionnee.

Les commandes differees selectionnees sont traitees en priorite par rapport aux commandes normales.

Ce traitement effectue les operations decrites dans les tables de decision qui suivent, ligne de commande par ligne de commande.

A- pour une commande enregistree

	cmde differee	cmde a facturer	etat du stock
stk epuise	0-› qte due	E-› indic. epuise 0-› qte livree qte cmdee-› qte restant due	
stk#epuise et qte en stk › ou = qte cmdee	0-› qte due	0-› qte restant due qte cmdee-› qte livree	qte en stk=qte en stk - qte livree
stk#epuise et qte en stk ‹qte cmdee	qte due= qte cmdee - qte en stk	qte livree = qte en stk qte restant due = qte cmdee - qte livree	0-› qte en stk qte diff=qte diff +(qte cmdee - qte livree)

Pour chaque produit commande, l'existence d'une quantite a livrer engendre une REQUISITION. Les requisitions relatives a une commande sont regroupees en une transaction appelee "bon de requisition". La date du jour figure dans un bon de requisition. On notera que si un produit est epuise, un message est enregistre dans le systeme.

B- pour une commande differee selectionnee.

	cmde differee	cmde a facturer	etat du stock
stk epuise	0-› qte due	E-› indic. epuise 0-› qte livree qte due -› qte restant due	
stk#epuise et qte en stk ›ou= qte due	0-› qte due	0-› qte restant due qte due -› qte livree	qte en stk=qte en stk - qte livree qte diff=qte diff - qte livree
stk#epuise et qte en stk ‹qte due	qte due= qte due - qte en stk	qte livree = qte en stk qte restant due = qte due - qte livree	0-› qte en stk qte diff=qte diff - qte livree (qte diff vaut 0 si elle est ‹ 0

Les regles de requisition, ainsi que le traitement des produits epuises sont analogues au cas de l'enregistrement d'une commande client. On notera que pour la mise a jour du stock:

- la ‹MAJ+› a priorite sur la ‹MAJ-›
- les commandes differees ont priorite sur les commandes normales.

3.5 PREPARATION DES COMMANDES (COLIS).

Les bons de requisition par commande, emis par le traitement ‹MAJ-›, sont accumules pour etre groupes par n <= 10 series de 10 commandes en un bon de requisition par serie, qui permet au magasinier de preparer au maximum 100 commandes - 10 * 10 - en optimisant son parcours au travers des rayons.

Pour preparer ces commandes, le magasinier dispose d'un chariot divise en 10 casiers dans lesquels il dispose les marchandises correspondant a 10 commandes.

En plus du bon de requisition par serie, le magasinier recoit un bon par casier qui specifie les 10 commandes affectees a ce casier. Ce bon par casier, il l'agrafe au casier correspondant lorqu'il commence son parcours en magasin.

BON DE REQUISITION PAR SERIE

	CASIER 1	CASIER 2	... ...	CAISER 10
No-produit, libelle	Qtte	Qtte	... ...	Qtte
... ...	...	...	... ...	...
... ...	...	...	... ...	...

BON PAR CASIER

No-casier, liste des Numeros de Commandes de ce casier.

3.6 EMBALLAGE ET FACTURATION

Le service d'emballage se compose de quelques stations equipees chacune d'un terminal a ecran et d'une imprimante - une seule imprimante pour toutes les stations - capable de lister tous les documents commandes par les differentes stations d'emballage.

Au fur et a mesure de l'arrivee des chariots, les preposes, a l'aide du bon par casier, connaissent les numeros des commandes preparees dans un casier. Chaque prepose, responsable d'une station d'emballage, affiche a l'ecran la commande qu'il a decide de traiter.

En puisant dans le casier et en suivant a l'ecran, il reconstitue le colis. S'il peut reconstituer ce colis entierement, il enclenche la facturation c'est-a-dire:

- l'impression d'une facture et d'un justificatif de non livraison
- l'impression d'une etiquette de colisage

- l'impression d'un bordereau SNCB ou Poste suivant le poids total du colis.

Cette facturation genere un mouvement comptable qui ira s'archiver sur le "fichier" des mouvements comptables.

S'il ne peut reconstituer le colis entierement, il met momentanement cette commande en suspens et poursuit avec les autres commandes du casier.

Quand toutes les commandes du casier ont ete passees en revue, il prend en charge la ou les commandes restees en suspens. Un magasinier, travaillant sous ses ordres, replace dans les rayons les marchandises restees dans le casier et, profitant de ce parcours en magasin, essaye de completer la ou les commandes incompletes.

Si le completage reussit totalement, le prepose a l'emballage se retrouve dans le cas normal et enclenche la facturation.

Si malgre ces recherches en rayons, le colis reste incomplet, il decide de l'envoyer tel quel au client mais de corriger a l'ecran les quantites reellement livrees.

Apres cette correction, il peut enclencher la facturation mais doit poursuivre par une serie d'operations rectificatrices:

- effectuer une mise a jour du stock pour le produit incompletement livre, c'est-a-dire:

    * soustraire de la Quantite en stock la quantite manquante tout en respectant la contrainte de non negativite du stock
    * ajouter a la Quantite differee, la difference entre la Quantite a livrer et la Quantite reellement livree.

- effectuer une mise a jour du "fichier" des Commandes differees par ajout d'un du

- avertir, par telephone, le gestionnaire des stocks, que certains stocks etaient inconsistants.

De toute facon, une transaction explicitant la modification effectuee est generee par l'ordinateur a destination du gestionnaire des stocks.

- ANNEXE 3 -

Présentation du problème de gestion du fichier des anciens étudiants de l'Institut d'Informatique.

On se propose d'automatiser la gestion du fichier des anciens étudiants de l'Institut d'Informatique.

1. Les informations qu'on désire - au minimum ! - voir figurer dans ce fichier sont reprises dans la fiche signalétique (cfr p.4.2 et 4.3);
2. Les opérations qu'on se propose de faire sur ce fichier sont
   - 2.1. Les opérations classiques de consultation et de mise à jour :
     - création d'un enregistrement
     - modification d'un enregistrement
     - suppression d'un enregistrement
     - consultation d'un enregistrement
   - 2.2. La production de divers états statistiques :
     - production d'étiquettes auto-collantes, avec possibilité de sélection par année et par orientation;
     - liste alphabétique des anciens par année d'obtention de diplôme et par orientation (cfr p.4.4); on peut ne souhaiter qu'une année particulière et/ou une orientation particulière;
     - tableau récapitulatif ventilant les anciens par année d'obtention de diplôme et par orientation (cfr p.4.5); même possibilité de sélection que pour la liste précédente.

SERVICE DES ANCIENS
INSTITUT D'INFORMATIQUE
rue Grandgagnage 21
5000 NAMUR

FICHE SIGNALETIQUE

A. Renseignements personnels

NOM (Majuscules) | | | | | | | | | | | | | | |

INITIALES DU PRENOM (Majuscules) | | | | | |

SEXE (Majuscule) | | (M = Masculin, F = Féminin)

ETAT-CIVIL (Majuscule) | | (M = Marié, C = Célibataire)

ANNEE D'OBTENTION DU DIPLOME DELIVRE PAR L'INSTITUT | | |

ORIENTATION SUIVIE | | (G = Gestion, S = Scientifique)

ADRESSE : Résidence ou villa (Majuscules), Bte.

| | | | | | | | | | | | | | | | | | | | |

RUE ET NUMERO (Majuscules)

| | | | | | | | | | | | | | | | | | | | |

CODE POSTAL

LOCALITE (Majuscules)

PAYS (Majuscules)

NUMERO DE TELEPHONE : PREFIXE NUMERO
(éventuel)

B. Renseignements professionnels

NOM DE L'EMPLOYEUR (Majuscules)

FONCTION EXERCEE (Majuscules)

ADRESSE DE L'EMPLOYEUR : RUE ET NUMERO (Majuscules)

CODE POSTAL LOCALITE

PAYS

TELEPHOENE : PREFIXE NUMERO EXTENSION

LISTE ALPHABETIQUE DES ANCIENS PAR ANNEE ET ORIENTATION.

ANNEE ACADEMIQUE : 1973

ORIENTATION : GESTION

DURAND, A.	Chaussée d'Ixelles, 156 1050 - BRUXELLES	02/3847245
FERON, G.	Avenue des anciens, 4, Bte 5 5100 - JAMBES	081/663399

.
.
.

ORIENTATION : SCIENTIFIQUE

.
.
.

Saut de page

TABLEAU RECAPITULATIF DES ANCIENS PAR ANNEE ET ORIENTATION.

ORIENTATION / ANNEE	GESTION	SCIENTIFIQUE	TOTAL
1973	3	2	5
1974			
1975			
1976			
1977			
1978			
1979			
TOTAL	64	18	82